NPSEC-93-017

NAVAL POSTGRADUATE SCHOOL

Monterey, California

SPC Toolbox: An Interactive MATLAB™ Package for Signal Modeling and Analysis, and Communications

by

Dennis W. Brown
and
Monique P. Fargues

October 15, 1993

Approved for public release; distribution unlimited.

Prepared for: Naval Postgraduate School
Monterey, CA 93943

Naval Postgraduate School

Monterey, California 93943-5000

Rear Admiral T. A. Mercer
Superintendent

H. Shull
Provost

This report was prepared in conjunction with classwork and research conducted at the Naval Postgraduate School

This report was prepared by:

Computer Engineering

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 15, 1993	

4. TITLE AND SUBTITLE

SPC Toolbox: An Interactive MATLAB™ Package for Signal Modeling and Analysis and Communications

5. FUNDING NUMBERS

6. AUTHOR(S)

Dennis W. Brown and Monique P. Fargues

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NPSEC-93-017

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this report are those of the authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release;
distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report presents the Signal Processing and Communications (SPC) software package. SPC is an interactive package designed to provide the user with a series of data manipulation tools which use MATLAB™ version 4 graphical interface controls. SPC includes various filtering techniques, AutoRegressive (AR) and linear Moving Average AutoRegressive (ARMA) modeling methods, speech processing and communication functions. SPC can be used in the classroom to illustrate and to reinforce basic concepts in signal processing and communications. It allows the user to concentrate on the principles presented in class instead of the details related to software usage. It can also be used as a basic analysis and modeling tool for research in signal processing. SPC was designed for Electrical Engineering applications. As a result, it is well suited to reinforce basic concepts presented in the following courses offered at the Naval Postgraduate School: EC 4410: Speech Processing, EC 4420: Modern Spectral Estimation, EC 3420: Statistical Digital Signal Processing, EC 3400: Digital Signal Processing, EC 2500: Communication Theory.

We hope that users will find this package useful, and we welcome any comments and suggestions regarding this software at browndw@ece.nps.navy.mil (until 6/94), or fargues@ece.nps.navy.mil.

14. SUBJECT TERMS

AR modeling, ARMA modeling, filtering, signal processing, communications, speech processing, engineering software, educational software

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

SPC Toolbox

An Interactive Matlab Package for
Signal Modeling and Analysis
and Communications
(with Speech Analysis
and
Linear Systems Modeling)

Technical Report no. NPSEC-93-017

Dennis W. Brown and Monique P. Fargues
Department of Electrical and
Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5121
October 15, 1993

Preface

The SPC (Signal Processing & Communications) software package is the result of work originated in Spring 1993 at the Naval Postgraduate School as a class project in speech processing. The project developed an interactive, user-friendly, tool to analyze speech signals using MATLAB. Then, as with any project left without a fixed deadline, it grew to include more sophisticated analysis tools such as various filtering techniques, Autoregressive (AR) and linear Auto Regressive-Moving Average (ARMA) modeling methods. Finally, it was expanded to include basic communication tools.

The final product is a software package designed to provide the user with a series of data manipulation tools which use MATLAB v.4 graphical user interface controls. SPC can be used in the classroom to illustrate and to reinforce basic concepts in digital signal processing and communications. It frees the user from having to write and debug his/her own code and gives him/her more time to understand the advantages and drawbacks of each technique included in the package. It can also be used as a basic analysis and modeling tool for research in Signal Processing. SPC is well suited to reinforce basic concepts presented in the following courses offered at the Naval Postgraduate School:

EC4410: Speech Processing,
EC4420: Modern Spectral Estimation,
EC3420: Statistical Digital Signal Processing,
EC3400: Digital Signal Processing,
EC2500: Communication Theory.

We hope that users will find this package useful, and we welcome any comments and suggestions regarding this software at browndw@ece.nps.navy.mil (until 6/94), or fargues@ece.nps.navy.mil.

TABLE OF CONTENTS

Commands

Signal Generation:

antpodal	Generate a baseband, antipodal [-1,1] signal.
unipolar	Generate a baseband, unipolar [0,1] signal.
sawwave	Generate a baseband, unipolar [0,1] or antipodal [-1,1] sawtooth wave. This function can also be used to generate a ramp.
sqwave	Generate a baseband, unipolar [0,1] or antipodal [-1,1] square wave.
triwave	Generate a baseband, unipolar [0,1] or antipodal [-1,1] triangular wave.
coswave	Generate a sampled cosine wave.
sinwave	Generate a sampled sine wave.
bfsk	Generate a frequency-shift keyed, bandpass signal with a random message.
bfskmsg	Generate a frequency-shift keyed, bandpass signal with a specific message.
bpsk	Generate a phase-shift keyed, bandpass signal with a random message.
bpskmsg	Generate a phase-shift keyed, bandpass signal with a specific message.
ook	Generate an on-off keyed, bandpass signal with a random message.
ookmsg	Generate an on-off keyed, bandpass signal with a specific message.
setsnr	Mix two signals such that the output signal's total bandwidth signal-to-noise ratio is set to the specified value.
setsnrbw	Mix two signals such that the output signal's noise-equivalent (in-band) signal-to-noise ratio is set to the specified value.

Coding:

lrs	Generate a maximal-length, linear recursive sequence.
str2masc	Convert a string into its binary ASCII representation as generated by a modem.
masc2str	Convert a vector of 1's and 0's representing ASCII into a string.

Modulation:

dsbsc	Generate a double-sideband, suppressed-carrier amplitude modulated signal.
dsblc	Generate a double-sideband, large-carrier amplitude modulated signal.
modindex	Returns the modulation index of an amplitude modulated signal.
envelope	Returns the envelope of a amplitude modulated signal.

Filtering:

avsmooth	Smooth a curve using an average smoothing filter.
mdsmooth	Smooth a curve using a median smoothing filter.

Data Files:

ld8bit	Load a file of samples stored as 8-bit signed integers.
ld16bit	Dito for 16-bit signed integers.
ld82bit	Dito for 32-bit signed integers.
loadsspi	Load a datafile store as ASCII in the format specified by the Cyclic Spectral Analysis Software Package from Statistical Signal Processing, Inc.

loadvoc	Load vector from a Creative Labs Soundblaster Voice file.
savevoc	Save vector to a Creative Labs Soundblaster Voice file.

AR/ARMA Modeling:

ar_corr	Compute parameters for an AR model using the autocorrelation method.
ar_covar	Compute parameters for an AR model using the covariance method.
ar_mdcov	Compute parameters for an AR model using the modified covariance method.
ar_burg	Compute parameters for an AR model using the Burg method.
ar_prony	Compute parameters for an ARMA model using the Prony method.
ar_durbn	Compute parameters for an ARMA model using the Durbin method.
ar_shank	Compute parameters for an ARMA model using the Shank method.
ar_levin	Compute parameters for an AR model using the Levinson recursion.

Least Squares Filtering:

ls_whopf	Compute the filter coefficients for a least-squares optimal filter using the Wiener-Hopf equation.
ls_svd	Compute the filter coefficients for a least-squares optimal filter using singular value decomposition.

Linear Systems:

minphase	Return a polynomial that is in minimum-phase form.
maxphase	Return a polynomial that is in maximum-phase form.
normaleq	Solves a system of Normal equations used in linear predictive filtering.

Graphic Display:

plottime	Plot the time-domain display of a signal.
lperigrm	Display the periodogram of a signal on a logarithmic scale.
wperigrm	Display the periodogram of a signal on a linear scale.

Speech Processing:

sp_steng	Compute the short-time energy.
sp_stmag	Compute the short-time magnitude.
sp_stzcr	Compute the short-time zero-crossing rate.

Dodads:

vectedit	Interactive cut and paste vector editing.
vectfilt	Interactive filter design and filtering.
vectarma	Interactive AR and ARMA modeling.
vecttime	Interactive speech time-domain analysis.
sanalyzr	Spectrum analyzer.
grafiltr	Graphical filter design through individual (complex-conjugate pair) placement of poles and zeros on the pole-zero plot of the filter transfer function.

INTRODUCTION

The SPC Toolbox provides a number of commands implementing a variety of digital signal processing and communication techniques. All of the routines are implemented using Matlab (version 4.0) M-files. Most of the commands are stand-alone or rely on other SPC Toolbox routines. Some commands however, rely on commands from the Matlab Signal Processing Toolbox available from The MathWorks. The functions that rely on the Signal Processing Toolbox are routines that implement FIR windows, analog IIR filter prototypes, or some spectral estimates.

The SPC Toolbox commands can be grouped into the following areas:

- Sampled signal generation.
- Bandpass signal generation and modeling.
- Information signal coding.
- Communications systems.
- AR and ARMA modeling.
- Least squares filtering.
- Linear systems.
- Speech signal processing.
- Signal display functions.
- Formatted signal data file loading and saving.

Additionally, a number of graphics-based tools (referred to as "dodads" in this document for lack of a better name) provide an interactive, graphical environment offering relief from the Matlab command line prompt. Some dodads essentially act as "front-ends" from which other commands are called (i.e. the vector filtering dodad). Others provide interactive modeling facilities (i.e. AR/ARMA modeling dodad). Still, others, provide a graphical environment to perform tasks not readily available or easily performed using Matlab commands (i.e. visual vector editing). The following dodads are provided in the SPC Toolbox:

- Vector Editor Dodad (visual "cut and paste" vector editor, vectedit).
- Vector Filtering Dodad (interactive filtering, vectfilt).
- AR/ARMA Modeling Dodad (interactive modeling, vectarma).
- Speech Time-Domain Analysis Dodad.
- Spectrum Analyzer Dodad (spectral measurement).

SPC Toolbox Conventions

The following conventions are followed by all commands in the SPC Toolbox:

- All commands have been tested in Matlab version 4.0, running on Sun workstations or on PC's under Microsoft Windows. Although untested, SPC Toolbox commands should work with no problems on other platforms running Matlab version 4.0.

• Except for dodads, no commands are interactive (i.e. request input from the user via prompts).
• All commands taking vectors as input arguments accept either Nx1 or 1xN vectors.
• Vectors are returned as Nx1 vectors (except functions returning vectors representing polynomials which return 1xN vectors following the Matlab polynomial convention).
• Arguments returned after an error occurs are returned as empty vectors. This allows error checking by user implemented routines with the following code:

```
y = bpsk(50,10240;
if isempty(y),
        error("An error occurred generating a BPSK signal.");
end;
```

• The default sampling rate used is that of the Sun workstation speaker output equal to 8192 Hz. This allows signals to be heard just as if they were being heard from a radio speaker. PC's with soundcards also have this feature available, except that the output sampling frequency is not limited to 8192 Hz.
• On-line help is available by typing "help command_name" at the Matlab prompt. A description of all SPC Toolbox commands can be viewed by typing "help spctools".
• In the documentation, commands are printed in a sans-serif font and variable names are printed in *italics*. Commands in the table of contents are shown in capital letters. All examples should work as shown if the toolbox is properly installed.

Customizing

Most colors of objects within dodads can be customized to meet individual preferences by editing the file spcolors.m. On workstations, individual users can copy the spcolors.m file from the spctools directory and modify it to meet their needs. To make individual copies of spcolors.m take precedence over the default spcolors.m, the only requirement is that the individual's copy of spcolors.m is reachable in the matlabpath before the spctools directory. This can be accomplished by keeping a copy of spcolors.m in the current directory or by prepending the matlabpath with the directory containing the individual's copy of spcolors.m. See the Matlab User's Manual for information on the matlabpath variable.

In addition to colors, the default size and location of dodads may be specified in the spcolors.m file. Simply "uncomment" the lines according to the instructions in the spcolors.m file and replace the parameters for the location and size with the user's preference. The default screen sizes are nine tenth of the screen for screens less than 800 pixels wide, eight tenths of the screen for screens from 800 to less than 1024 pixels wide, seven tenths of the screen for screens 1024 pixels wide and five tenths of the screen for screens greater than 1024 pixels wide. The default size and location for child graphic windows cannot be specified.

The file spbandw.m contains the default colors for a black and white screen. Using these colors for the default requires executing the Matlab whitebg command before executing any dodad commands. To use this file, copy or rename to file to spcolors.m and follow the instructions above.

Error Messages

A considerable amount of error checking is performed in the SPC Toolbox routines in an attempt to avoid disasters. All error messages generated by SPC Toolbox routines have the format:

```
function_name: error message
```

If you receive an error message that does not have the name of a SPC Toolbox routine, the error message was generated by Matlab or a command from some other toolbox. Currently, the SPC Toolbox relies only on version 4.0 Matlab commands and a few commands from the Matlab Signal Processing Toolbox.

If you receive an error while executing a SPC Toolbox command, please send email to "browndw@ece.nps.navy.mil" (good until June 94) or "fargues@ece.nps.navy.mil". Include a description of the operations in progress when the error was received and a copy of the error message itself. If possible, cut the command and error message right from the shell used by Matlab and paste it into the email text using the X Windows (or MS Windows) cut-and-paste facilities.

Troubleshooting Hints

There are some limitations in using Matlab as an interactive, graphical tool due to limitations of the Matlab programming language. The following presents methods to recover from the most common problems.

• The graphical tools make extensive use of global variables. Therefore, only one graphical tool of each kind can be open at the same time in a Matlab session (i.e. two vectfilt dodads cannot be used at the same time).

• Care must be taken when using the plot command from the Matlab prompt when a dodad is open. The Matlab plot command uses the current axis in the current figure window for its output. If a dodad is the current figure when the plot command is used from the command line, the axis inside the dodad is used. The simplest way to recover from this problem is to close the dodad and to start over. To prevent this problem, open a new figure window using the figure command before using plot from the command prompt (in fact, before any other command that creates a plot).

• In the event an error occurs when using a dodad, try closing the dodad using the close pushbutton. If this fails, close any other open dodads and then execute the commands "close; clear global" to recover.

Matlab Signal Processing Toolbox Dependencies

The following SPC Toolbox commands depend upon various commands from the

Matlab Signal Processing Toolbox.

Table 1: Signal Processing Toolbox Dependencies

SPC Toolbox	Matlab Signal Processing Toolbox
vectfilt	butter, cheby1, cheby2, ellip, hamming, hanning, barlett, blackman, spectrum
vecttime sanalyzr sp_steng sp_stmag sp_stzcr	hamming, hanning, bartlett, blackman, triang, kaiser
envelope modindex	hilbert

Installation Under Matlab for Microsoft Windows

The following steps are required to install the SPC Toolbox on a personal computer with Matlab running under Microsoft Windows:

1. Create a directory called "SPCTOOLS" in the "\MATLAB\TOOLBOX" directory.

2. Copy the SPC Toolbox *.m files to the "\MATLAB\TOOLBOX\SPCTOOLS" directory.

3. Edit the "MATLABRC.M" file located in the "\MATLAB" directory. Find the section similar to the following example MATLABRC.M file and insert the line:

```
'drive_letter:\matlab\toolbox\spctools;',...
```

The location of this line can be critical if any command in any other toolboxes has the same name as a SPC Toolbox command. Care has been taken to name SPC Toolbox commands differently from those used in the Matlab Signal Processing Toolbox, Control Systems Toolbox or System Identification Toolbox as of 9/93. To ensure SPC Toolbox commands take precedence over any other toolbox commands of the same name, enter the above line near the top of the matlabpath specification. To ensure SPC Toolbox commands do not take precedence, enter the above line near or at the bottom of the matlabpath specification.

```
% Setup the MATLAB search path.
matlabpath([...
'e:\MATLAB;',...
'e:\MATLAB\toolbox\matlab\general;',...
'e:\MATLAB\toolbox\matlab\demos;',...
```

```
.....
'e:\matlab\toolbox\signal;',...
'e:\matlab\toolbox\ident;',...
'e:\matlab\toolbox\control;',...
'e:\matlab\toolbox\spctools;',...
'e:\matlab\myprogs;',...
]);
```

Installation on Sun Workstations/Networks

Toolbox Directory

1. Create a "spctools" directory in the "toolbox" directory.
2. Change to the spctools directory using the UNIX cd command.
3. To copy the commands from an MS-DOS formatted floppy disk, use the mtools command "mcopy -t 'a:*.m' ." (period inside the quotes is required). The "-t" option is required to convert the text file format of the m-files from MS-DOS text file format (cr/lf end-of-line markers) to UNIX text file format (lf end-of-line marker).
4. Add the "spctools" directory to the matlabpath.

Disclaimer

This software package is made available as a service to the academic community. It is not a depository of Mathworks approved software. The authors make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this report. Use of these programs is at the users' own risk and the authors disclaim all liability for injury, damage, and losses that may result from their use. Losses include, but are not limited to, loss of data or data being rendered inaccurate, or losses sustained by third parties for a failure of the software to operate. The U.S. Government assumes no responsibility for the information provided. Matlab is a trademark of The Mathworks. Microsoft Windows is a trademark of Microsoft. Sun is a trademark of Sun Microsystems. Soundblaster is a trademark of Creative Labs, Inc.

TUTORIAL

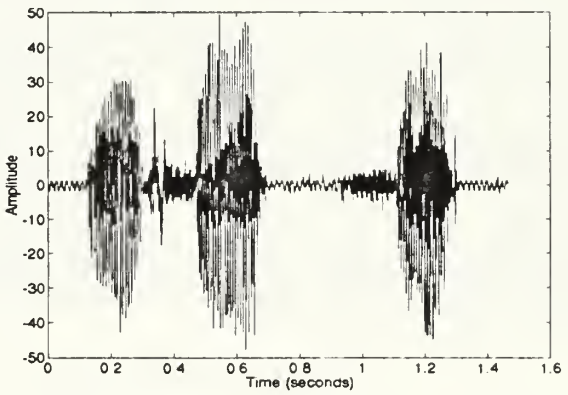
Vector Edit Dodad

This tutorial introduces the basic operation of the Vector Edit Dodad (vectedit). The Vector Edit Dodad provides an interactive, graphical environment within Matlab to edit vectors (naturally). This is ideally suited for editing vectors representing digitized signals such as speech. In fact, in this tutorial, we will use vectedit to rearrange the words in a speech signal.

A file named "seatsit.voc" is contained on the distribution disk or in the "spctools" directory. This file contains the words "the seat, sit" as spoken by the author. It was recorded on a PC using a microphone equipped Soundblaster audio board. The file is stored in the Soundblaster Creative Voice file format. To load the file into Matlab, the loadvoc command from the SPC Toolbox is used:

```
seatsit = loadvoc('seatsit');      % insert pathname as appropriate
```

Start the Vector Edit Dodad with the command "vectedit(seatsit)". Once loaded, the plot in the vectedit dodad window will look like the following.

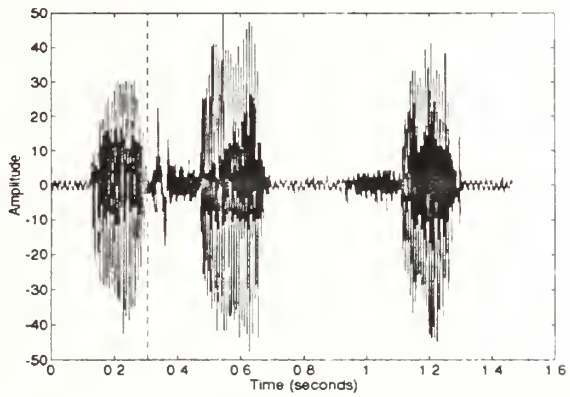


First, note that a 60 Hz interference can easily be seen during periods of silence. Second, note the /s/ section of the word "seat" contains a large, low-frequency transient. Third, note that the signal was sampled at 8192 Hz and can be sent to the audio output of a Sun workstation (or soundcard equipped PC) by using the "Play All" popupmenu item.

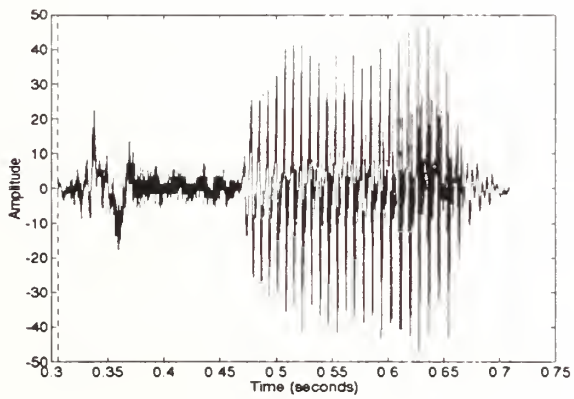
One useful function of vectedit (even when the user doesn't want to actually edit a vector) is to use vectedit as an inspection tool. The user can simply load a vector into vectedit and use the zoom function to get a clearer picture of a particular feature contained in the vector. The user could then take a "snapshot" of the zoomed portion as a preliminary means of obtaining a hardcopy. This method replaces the hit-and-miss process of guessing the correct subscripts to use with the Matlab plot command. In this example, we will use the vectedit dodad to take a better look at the word "seat."

To zoom in on the word "seat," the first step required is to set the start and stop markers to surround the word. Place the mouse cursor over the "Mark Start" pushbutton and press the left mouse button. At this time, the cursor changes into a cross-hair and the plot title changes to "Mark beginning with cursor." Move the cursor until it is somewhere above the small "valley" immediately after the first large "burst" as depicted by the dashed line in the figure below. When in position, press the left mouse button. Once the mouse button is released, the cursor changes back into an arrow and the title is cleared. A vertical dashed line appears at the point on the time axis where the mouse button was pressed. The dashed line is the beginning marker. Note a dotted line appears at the end of the signal (not shown). This is the end marker and since it has not been placed yet, it is still at the end of the signal.

Using a similar procedure, place the end marker as shown below.

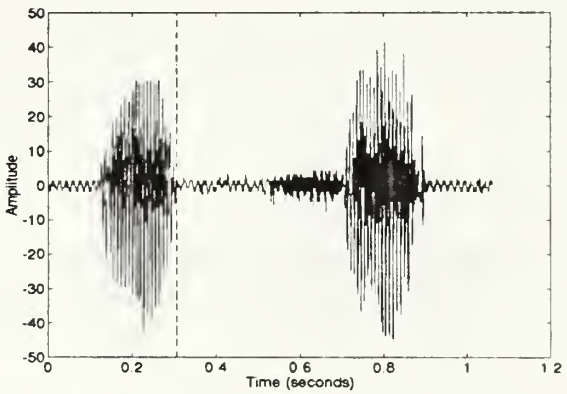


Now that the word "seat" has been marked, all menu selections that apply to ".. Marked" will apply to this portion of the signal. Selecting "Play Marked" plays the word "seat" only. To get a better look at the word "seat," select "Zoom Marked" from the Zoom pop-upmenu.

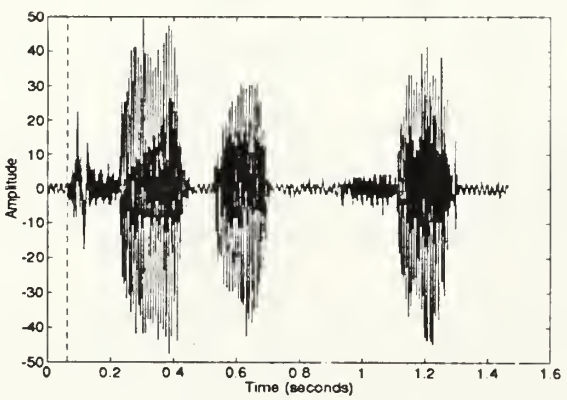


The low frequency transient can be seen more clearly now. The user can reposition the markers around the transient and select "Zoom Marked" once again to get an ever closer look. Once you are satisfied with the capabilities the zoom feature, select "Zoom Full" to view the whole signal and to reposition the markers to surround the word "seat."

In reordering the words, we "cut" the word "seat" and then "paste" it in before the word "the." With the markers surrounding the word "seat," move the cursor over the "Cut" pushbutton and press the left mouse button. Selecting the cut pushbutton removes the portion of the signal between the two markers and stores it in a cut-and-paste buffer. It then collapses the signal and redraws it appropriately. Both markers are now located in the same spot as shown below. As long as both markers remain in this spot, selecting the cut pushbutton again would reinsert the cut portion back into the same location. This "undo" feature is only valid as long as the markers are not moved. Note that selecting the cut pushbutton has no effect when the markers are located at the beginning and end of the vector (as it was originally the case), .



The remaining step consists in pasting the word "seat" into the signal before the word "the." Move the cursor to the "Paste" pushbutton and press the left mouse button. After selection, the cursor changes into a cross-hair and the plot title changes to "Mark insertion point with cursor." Move the cursor to the period of silence before the word "the" and press the left mouse button. The word "seat" is now the first word in the signal. Selecting "Play Full" will confirm this fact.



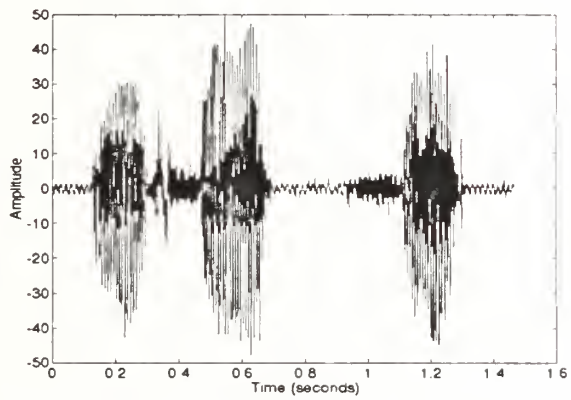
See Common Controls and VECTEDIT for further discussion on the use of the vectedit dodad.

TUTORIAL

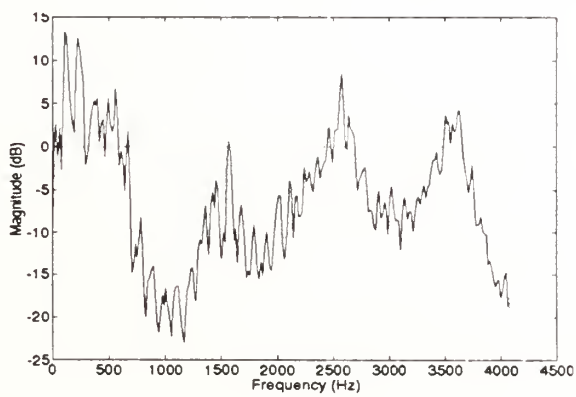
Vector Filter Dodad

This tutorial introduces the basic operation of the Vector Filter Dodad (vectfilt). The Vector Filter Dodad provides a graphical, interactive environment to design filters and to apply the filter to a signal. Note that vectfilt has the capability to display the filter transfer function and the spectrum of the signal the filter is to be applied to simultaneously. Furthermore, the filter cutoff frequencies can be selected with the mouse.

In this tutorial, we design a Chebychev Type II bandpass filter to study the formant frequencies contained in a speech signal. Load the "seatsit.voc" file as discussed in the vectedit tutorial and start the vectfilt dodad with the command vectfilt(seat). Once loaded, the plot should look like the following.

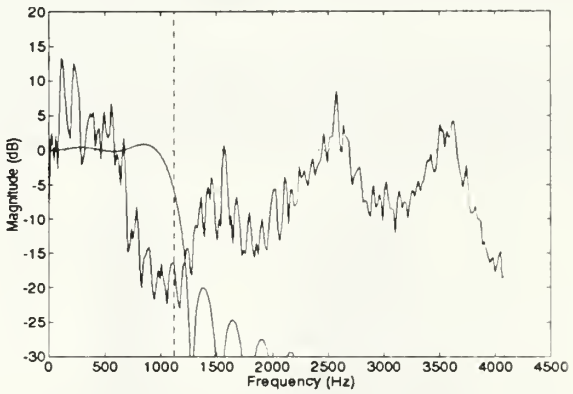


Both time and frequency domains are available in the vectfilt dodad. As could be expected, selecting the "Time" radiobutton displays the time-domain (as shown above) and selecting the "Spectrum" radiobutton displays the frequency domain (as shown below). The spectrum shown is a power spectrum estimate across the entire length of the signal using the Welch method with a 1024-point window. See the Matlab Signal Processing Toolbox spectrum command for further information. Only the positive half of the frequency spectrum is displayed, thus, the frequency resolution is half the sampling frequency divided by 512. The spectrum is normalized as discussed in the spectrum command description..

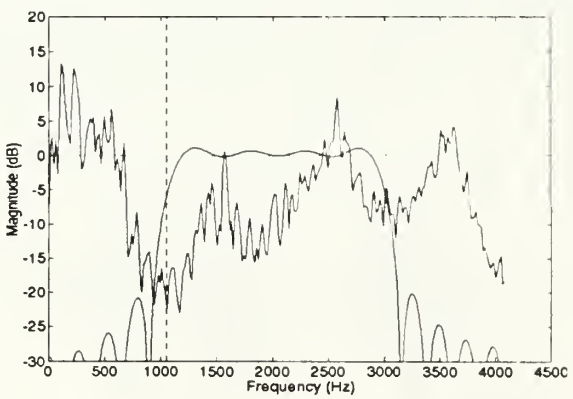


When in the frequency domain, both linear and logarithmic scaling are available by selecting the so named radiobuttons. Also, it is not necessary to be in the frequency domain to design filters. All controls can be set and the cutoff frequencies specified using the edit boxes while in the time domain. However, design while in the frequency domain does have its obvious advantages.

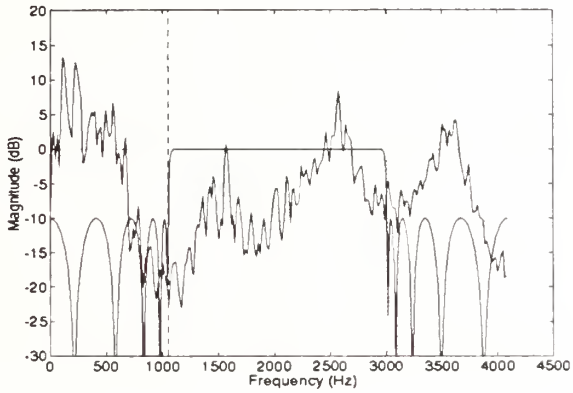
In filter design, the begin marker marks the lower cutoff frequency and the end marker marks the upper cutoff frequency. Place the lower cutoff frequency in the valley between the first and second formants as shown below (about 1200 Hz).



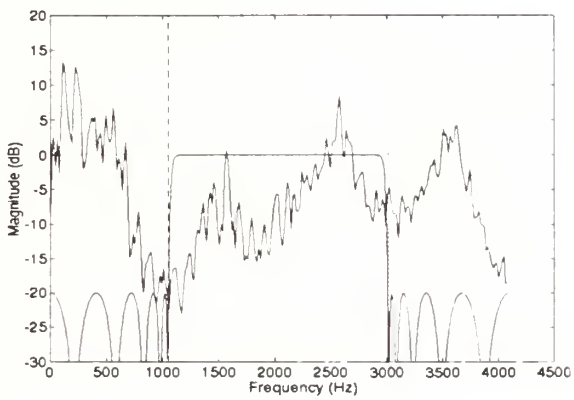
Once a cutoff frequency has been set, the transfer function of the filter specified by current settings is displayed. In the case shown, a lowpass FIR filter was specified by the controls settings at the time the cutoff frequency was set. Next, place the end marker in the valley between the third and fourth formants (about 3000 Hz) and select "Bandpass" from the filter type popupmenu. This produces the following FIR bandpass filter.



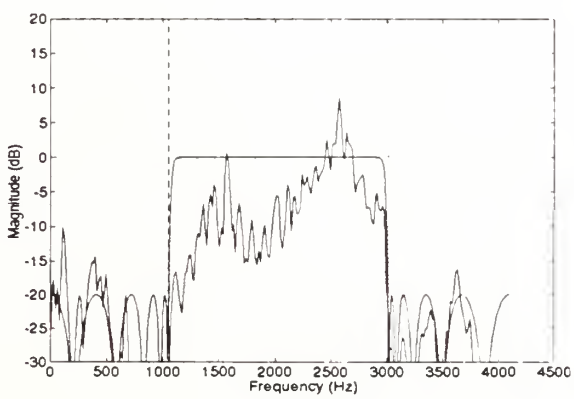
Use the filter popupmenu to select a Chebychev Type I filter which is characterized by a flat passband, steep cutoff and an equal ripple in the stopband (switching to a linear display better shows the equal ripple).



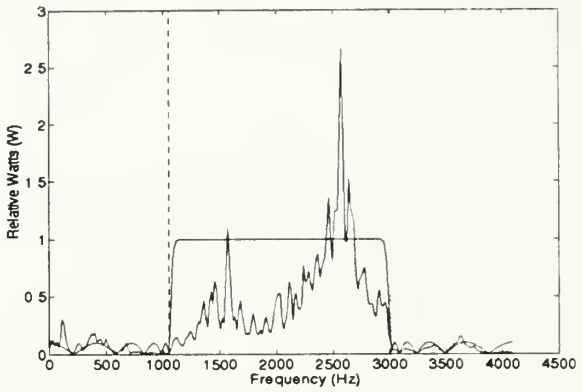
The default attenuation in the stopband is 10 dB. Using the stopband attenuation popupmenu, change the stopband attenuation to 20 dB.



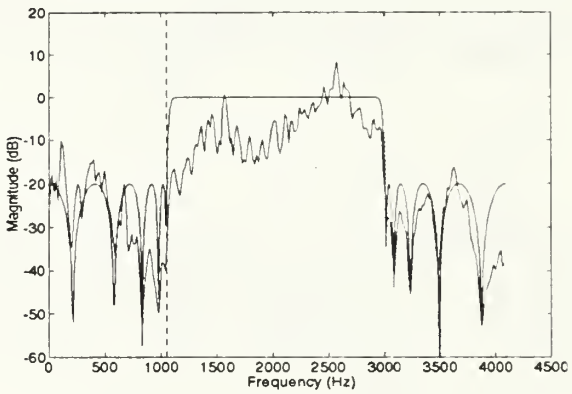
The filter design is now complete. Select the apply pushbutton to filter the signal. After the signal has been filtered, its spectrum is recomputed and redisplayed along with the filter transfer function. In the result shown below, the filter appears to have performed quite well.



If you have not done so, switch over to a linear scale for a quick look



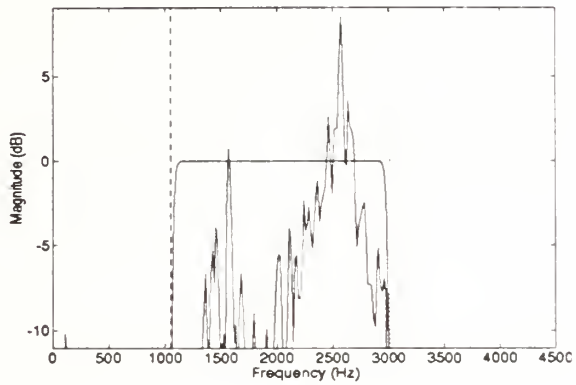
Now switch back to the logarithmic scale.



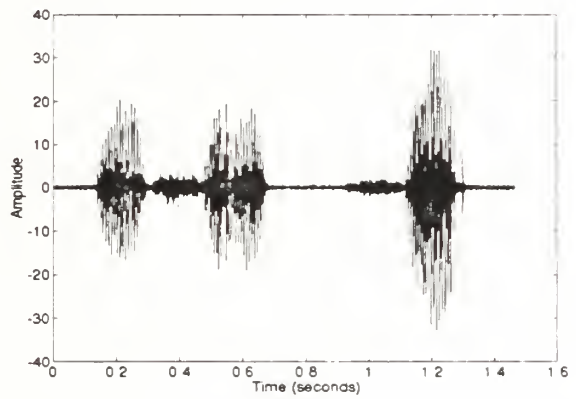
Note the difference in the display. What has occurred is that when the scaling was changed and the plot redrawn, the limits on the plot axis are set such that the entire magnitude of the spectrum is visible. After the signal was filtered, the magnitude of the frequencies corresponding to the nulls in the transfer function became very small and, as such, require a larger dB axis to display. You might wonder why this axis change did not occur immediately after the filter was applied. The answer is the axis was held to its pre-filtered state to allow easier comparison of before and after effects of the filter.

Sometimes nulls will extend so far down as to make it difficult to see details of the spectrum around 0 dB. Fortunately, the magnitude axis can easily be changed to effectively zoom in on 0 dB. Unlike zooming in on the time or frequency (horizontal) axis, the magnitude (vertical) axis can be zoomed in by simply clicking the mouse with the cursor somewhere inside the axis. On a logarithmic scale, clicking the mouse with the cursor below zero resets the lower axis limit to the magnitude the cursor was at when the mouse was clicked. The same operation with the cursor above 0 dB sets the upper axis limit. On a linear scale, any mouse click with the cursor inside the plot axis resets the upper axis limit. Note that the cursor must be in a blank portion of the screen within the axis when using this method (not near any curves or other lines).

The following is the result of zooming in on 0 dB using the above method.



Switching back to the time domain illustrates the filter effect. Note that the 60 Hz interference and the low frequency transient have been removed. Playing the signal using the play popupmenu will demonstrate the audible contribution the f2 and f3 formant frequencies make to this sample of speech.



The restore pushbutton will reload the original signal allowing further experimentation.

As with vectedit, a "zoom" feature is supported by setting the beginning and end markers and selecting "Zoom Marked" from the zoom popupmenu. In vectfilt, this feature has been extended to the frequency domain. Markers in the time and frequency domains are independent and thus, when the display is zoomed in on a feature in the time domain, the same feature is still zoomed in on after switching to the frequency domain and back.

See Common Controls and VECTFILT for further discussion on the use of the vectfilt dodad.

TUTORIAL

AR/ARMA Modeling Dodad

This tutorial introduces the basic operation of the AR/ARMA Modeling Dodad (vectarma). The AR/ARMA Modeling Dodad provides a graphical, interactive environment for:

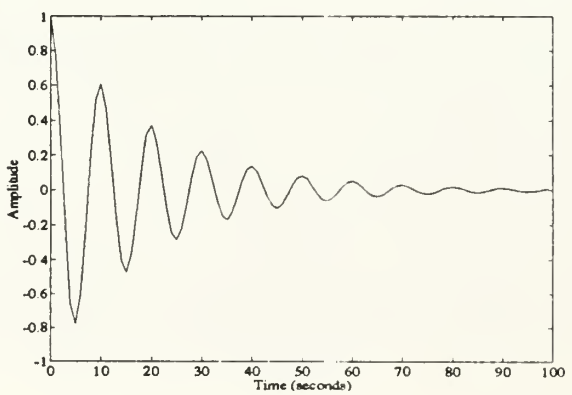
- Autoregressive (AR) modeling via the autocorrelation, covariance, modified-covariance, and Burg methods.
- Autoregressive-Moving Average (ARMA) modeling via the Prony, Durbin, or Shank methods.
- Performing spectral analysis using the above AR/ARMA modeling methods.

In this tutorial, we will:

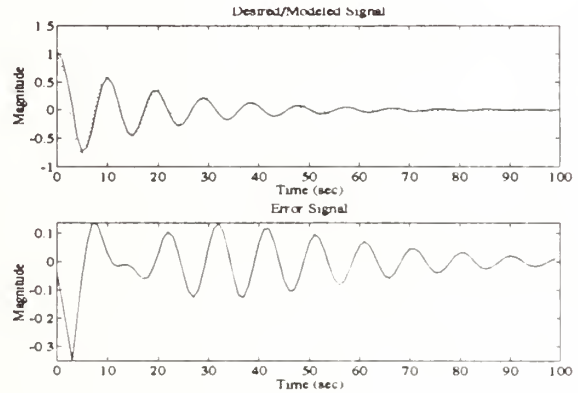
- Create an AR model using all available data.
- Create an ARMA model.
- Create an AR model using a portion of the available data.
- Create an ARMA model of a speech phoneme and then reconstruct the phoneme from the model.
- Perform spectral analysis on a speech signal.

Contained on the distribution disk (or in the spctools directory), are three files named seatsit.voc, uuu.mat, and t0x.mat which will be used in the tutorial. Each will have to be loaded into the matlab workspace before using. The file seatsit.voc is loaded using the SPC Toolbox command "seatsit = loadvoc('seatsit');" and the two *.mat files are loaded using the Matlab commands "load uuu" and "load t0x". The seatsit.voc file contains the words "the seat, sit" as spoken by the author, the uuu.mat contains the voiced vowel /u/, and the file t01 contains data sets T01, T02, T03 and T04 from [1].

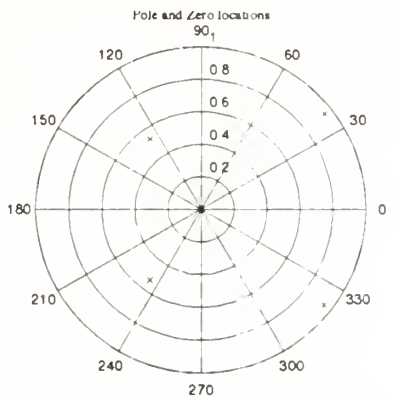
Start the AR/ARMA Modeling dodad using the command "vectarma(t01,1)". The second argument tells vectfilt to start with the sampling frequency set to 1 vice the default of 8192 Hz. When the sampling frequency is set to 1, the time scale shows the actual indices of the vector being displayed. The following plot should be seen in vectarma.



Three sets of marks are available in vectarma. During this first example, we will leave the marks in their initial position at the beginning and end of the vector. From the AR order popupmenu, Q, select a model order of four. Also, unselect the spectral analysis plot by unchecking the Spectrum checkbox. Leave all other settings to their default values and apply the model by selecting the Apply pushbutton. This will generate a fourth-order AR model using the autocorrelation method. After the model has been created, two graphic windows open containing the following two plots.



The upper plot in the first graphic window is the model (the solid line) overlayed on the desired data (the dotted line). The lower plot is the error obtained by subtracting the model from the desire data.



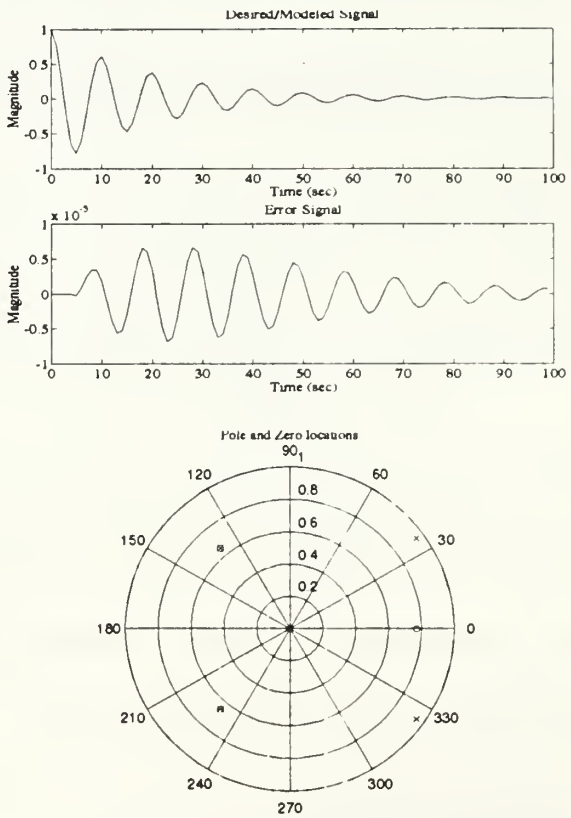
The second graphic window contains a polar plot of the pole locations.

The Matlab command window will contain the model data:

```
------------------------------------------
AR Model - Autocorrelation Method
------------------------------------------
Poles
  0.7479 + 0.5865i
  0.7479 - 0.5865i
 -0.3110 + 0.4334i
 -0.3110 - 0.4334i
```

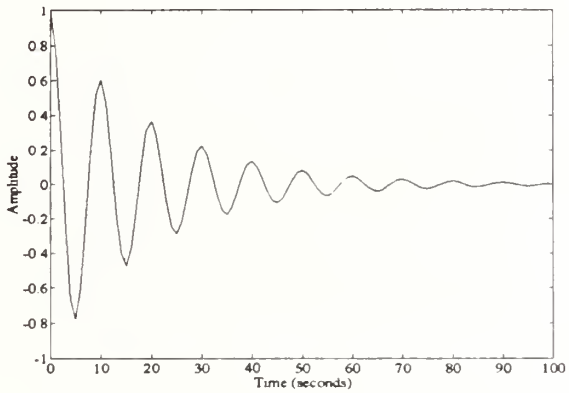
```
Magnitude  Angle(D)
  0.9504   38.1002
  0.9504  -38.1002
  0.5335  125.6648
  0.5335 -125.6648

Model coefficients:
   1.0000
  -0.8738
   0.2574
   0.1362
   0.2571

Sum-of-squared errors = 1.08
```

Next, we will generate an ARMA model using the Prony method. From the Model Type popupmenu select ARMA and then set the MA model order, Q, to four and apply. From the model/polar plots shown below, it can be seen this method much more accurately models the data set T01.

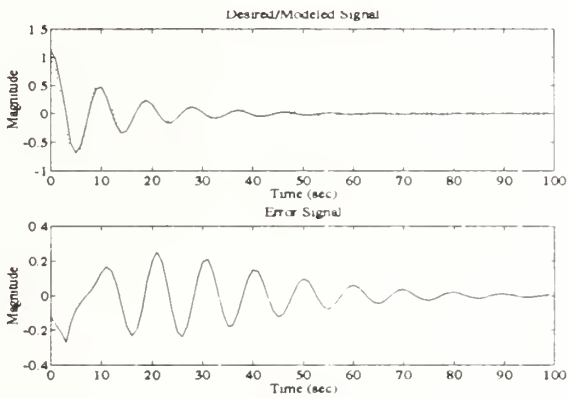


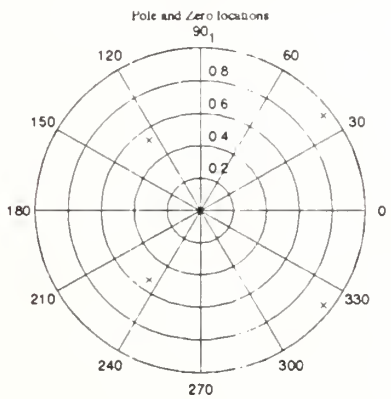
Sometimes it is not necessary to use all available data to sufficiently model an AR/ARMA process. Return to AR modeling by selecting AR from the Model Type popupmenu. Located at the bottom of the vectarma dodad are pushbuttons to set the Model Begin and Model End marks. These marks control data length used in generating the model. Until now, they have been set to the beginning and end of the entire data set. Using the Model End pushbutton, set the Model End mark to the bottom of the second cycle of the data as

shown below. When the model is applied this time, only the data between the start of the data (where the Model Begin mark is still located) and the Model End mark will be used in computing the model parameters. Apply the model at this time.



Shown below are the results of using only this first portion of data. Compare these results to those obtained in the first example.

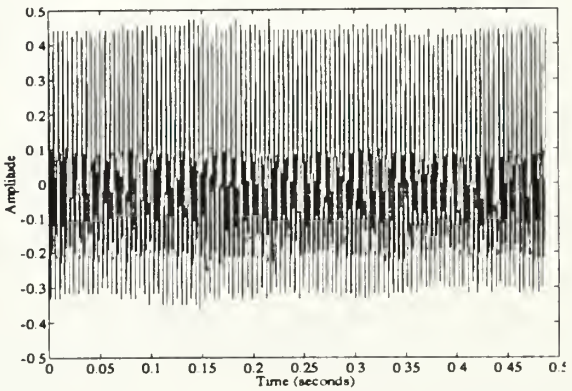




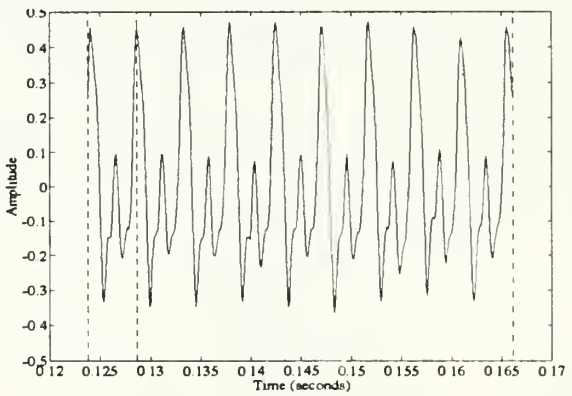
To continue on with this tutorial, the data from the file uuu.dat has to be loaded into vectarma. This can be accomplished by either closing the vectarma dodad and restarting it with the command "vectarma(uuu)" or by selecting the Load pushbutton and entering "uuu" into the edit box which temporarily replaces the Save and Load pushbuttons. To load uuu by this method, uuu has to exist already within the Matlab workspace and the edit box has

to be selected with the mouse before the name can typed into it. If you load uuu using the Load pushbutton, change the sampling frequency to 8192 Hz using the popupmenu.

The following plot should now be displayed in vectarma.



Use the Begin and End marks to zoom in on at least eight cycles as shown below. To prepare for this example, set the Model Begin mark to the highest peak of the first whole cycle and set the Model End mark to the same point seven full cycles away from the Model Begin mark. Set the Period End mark to the peak of the cycle immediately after the Model Begin mark as shown.

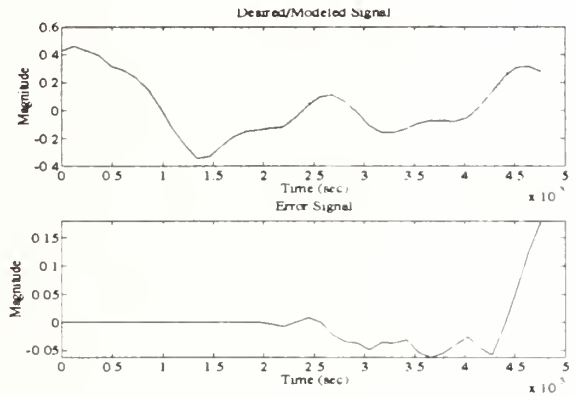


We are about to do the opposite of what was done in the previous example. In that example, a model was generated that was longer than the data used. In this example, we will generate a modelled signal that is shorter than the data used to generate the model parameters. The period markers are used to set the length of the modelled signal to be generated once the model coefficients are computed. When the Model Begin mark is set, the Period Begin mark is set to the same location. This is usually how these marks are used although the Period Begin mark can be set AFTER setting the Model Begin mark. The distance between the Period marks is displayed in the Chain Period edit box. In this example, the Chain Period represents the pitch period of the voiced phoneme.

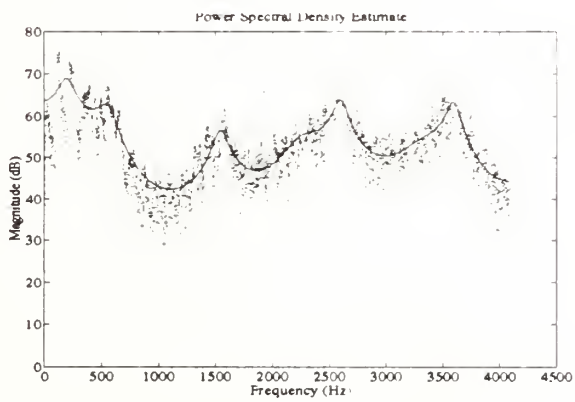
In order to reconstruct the phoneme from the model, several periods of the model will have to be "chained" together. Check the Chain checkbox and select a count of 50 periods

from the Chain Length popupmenu. When the model is applied, 50 periods of the model are chained together for form the reconstructed phoneme. This reconstruction can be audibly compared to the original by selecting Play Desired and Play Model from the Play popupmenu.

Set vectfilt up to generate an ARMA model using the Prony method with 16 poles and 16 zeros. The order 16 is specified by selecting the User option from the Q and P popupmenus and by entering "16" into the edit box that temporarily replaces each popupmenu. The following shows the original signal, the modelled signal and the error signal obtained after applying the model.

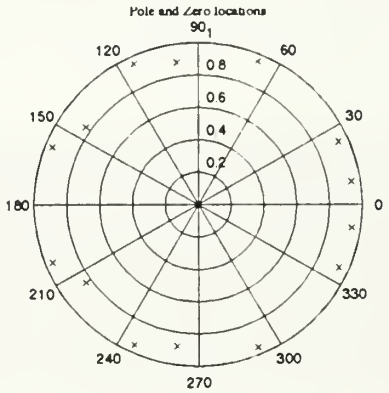


The last example uses an AR model to estimate the frequency content of seatsit. Load seatsit into vectarma by one of the methods previously suggested for loading uuu. Set the controls in vectarma to generate a 14th order AR model using the autocorrelation method. Be sure to turn the chain off and turn the spectrum plot on. Also, you might want to turn off the overlay and error plot options as they are of no use in this example. Applying this n[illegible] results in the following power spectral density estimate.



In this plot, the scattered dots represent the values of the fourier transform of the signal. The solid line is the transfer function specified by the model coefficients and is plotted using the freqz function from the Matlab Signal Processing Toolbox. The peak frequencies can be determined by comparing the power spectral density plot to the polar plot and the

model parameters displayed in the Matlab command window.



The user is encouraged to repeat this tutorial using different models to study their effects.

Recommended AR/ARMA modeling and spectral estimation references:

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, Prentice-Hall, 1992.

[2] William A. Gardner, *Statistical Spectral Analysis, A Nonprobabilistic Theory*, Prentice-Hall, 1988.

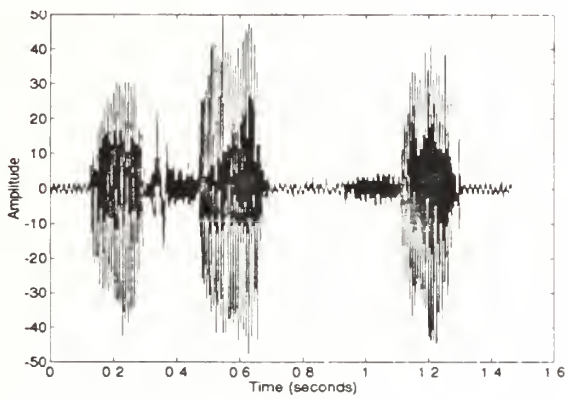
[3] Steven M. Kay, *Modern Spectral Estimation*, Prentice-Hall, 1988.

TUTORIAL

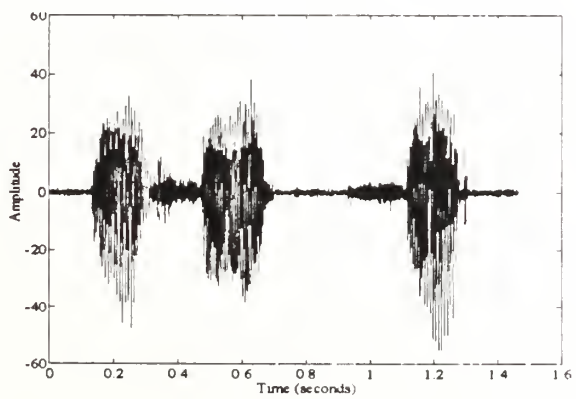
Speech Time-Domain Analysis Dodad

This tutorial introduces the basic operation of the Speech Time-Domain Analysis Dodad (vecttime). The Speech Time-Domain Analysis Dodad provides a graphical, interactive environment to apply time-domain methods to determine voice and unvoiced phonemes in speech. Time-domain methods available in the SPC Toolbox are the short-time energy, short-time magnitude and the short-time zero crossings methods. Voiced phonemes are characterized by a relatively high energy content while unvoiced phonemes are characterized by higher frequency content (hence, a higher number of zero crossings).

In this tutorial, we determine the voiced and unvoiced phonemes in the phrase "the seat, sit" contained in the file seatsit.voc. Load the file seatsit.voc as described in the vectarma tutorial and start the Speech Time-Domain Analysis Dodad with the command "vecttime(seatsit)". Note once again, the 60 Hz interference contained in this speech sample. To accurately apply time-domain analysis to this signal, the 60 Hz interference needs to be removed.

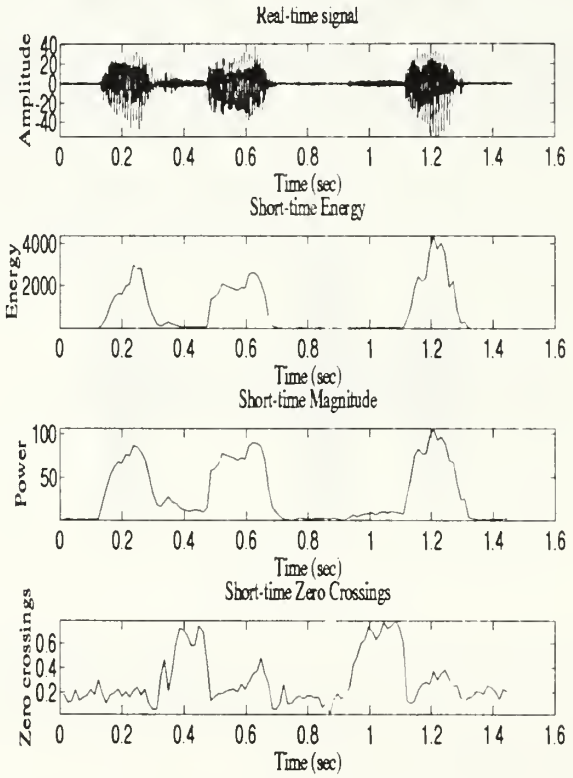


Without closing vecttime, start the vector filtering dodad using the command "vectfilt (seatsit)". Design a 10th order, highpass, Chebychev Type I filter with a lower cutoff frequency of approximately 80 Hz and apply the filter. This will result in the removal of the 60 Hz interference as shown below.

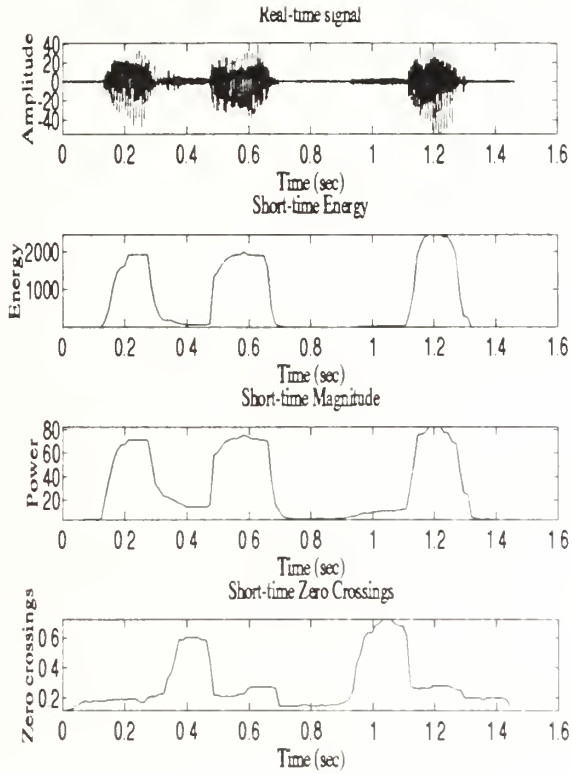


Without closing either the vectfilt or vecttime dodads, load the filtered signal now contained in vectfilt, into vecttime. This is accomplished by selecting the Common pushbutton in vecttime. The Common pushbutton is used to load the latest change made to a signal by either the vectfilt or vectedit dodads into any of the dodads in the SPC Toolbox without having to save the vector first and then starting a dodad with the saved vector as the input argument.

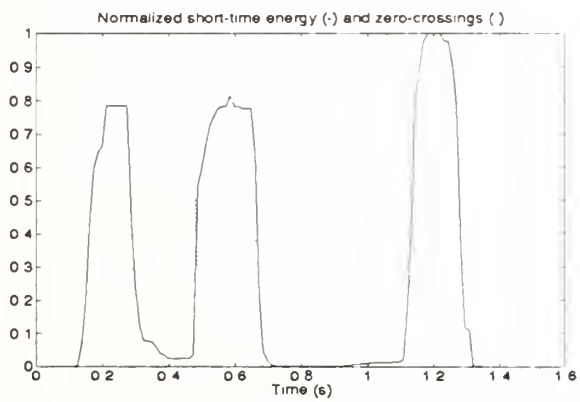
With the "clean" signal now in vecttime, check the Include Signal, Energy, Magnitude and Zero Crossings checkboxes. Also set the frame length to 15 milliseconds and the frame overlap to 20% and then apply. A long Matlab graphics window opens with the following plots of the analysis results. Regions with high energy or magnitude correspond to voiced phonemes in the real-time signal and the Zero-Crossing correspond to unvoiced phonemes.



Two smoothing methods are available to enhance the curves resulting from the analysis. Ideally, sharp transitions in the curves more clearly define the beginning and end of the phonemes represented. Select a Median filter from the Smoothing popupmenu and apply. Notice that the seperation between voiced and unvoiced phonemes is more clearly seen after applying the median filter.



Further manual analysis can be used to combine the output of either the short-time energy or magnitude analysis with the short-time zero-crossings analysis. Save the analysis output to a variable with the name "ss" using the Save pushbutton. This results in an Nx4 matrix being saved to the Matlab workspace in which the first column is the time scale, the second column is the short-time energy analysis curve, the third column is the short-time magnitude analysis curve and the fourth column is the short-time zero-crossings analysis curve. Use the following commands to normalize the short-time energy and zero-crossing curves and combine them on the same graph.




```
ssm = max(ss(:,2));
ssz = max(ss(:,4));
t = ss(:,1);
plot(t,ss(:,2)/ssm,'y',t,ss(:,4)/ssz,':');
```

```
title('Normalized short-time energy (-) and zero-crossings (:)');
xlabel('Time (s)');
```

Note the period of silence between the words "the" and "seat" is clearly shown.

Recommended speech signal processing references:

[1] John R. Deller, Jr, John G. Proakis, and John H. L. Hansen, *Discrete-Time Processing of Speech Signals*, Macmillian, 1993.
[2] F. J. Owens, *Signal Processing of Speech*, McGraw-Hill, 1993.
[3] L. R. Rabiner and R. W. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, 1978.

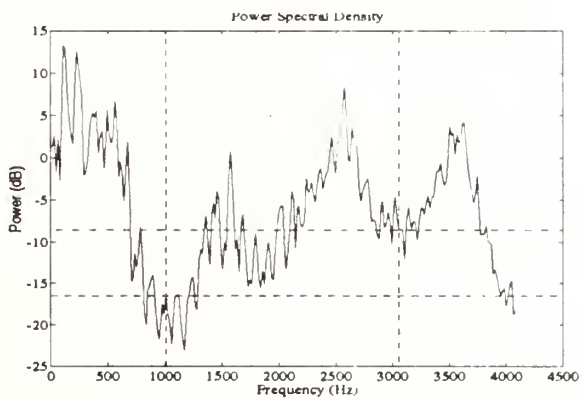
TUTORIAL

Spectrum Analyzer Dodad

This tutorial introduces the basic operation of the Spectrum Analyzer Dodad (sanalyzr). The Spectrum Analyzer Dodad provides a graphical, interactive environment designed to compute a signal's spectral components. Its operation is similar to that of the classical spectrum analyzer. Two independent cursors are provided along with a digital readout of their corresponding frequencies and magnitudes and the difference in frequency and magnitude between them. The cursors may be moved from sample-to-sample using the "<" or ">" pushbuttons or from peak-to-peak (high or low) using the "<<" or ">>" pushbuttons. Single "clicking" on the spectral line with the mouse cursor moves the closest spectrum analyzer cursor to the position of the mouse cursor. A spectrum analyzer cursor can also be grabbed with the mouse and dragged to a new position. The spectrum analyzer cursors can also be moved to an specific frequency be entering the frequency in the cursor's frequency edit box and pressing return.

Note: Movement of the spectrum analyzer cursors with the mouse under the Sun operating system using Openwindows requires holding the mouse button down until the cursor movement is completed. Also, after dragging a cursor with the mouse, the cursor must be held down until the cursor position stabilizes. Under Microsoft Windows, a quick click is all that is required to move the closest spectrum analyzer cursor to the mouse cursor location and no pause is required after dragging a spectrum analyzer cursor to a new position.

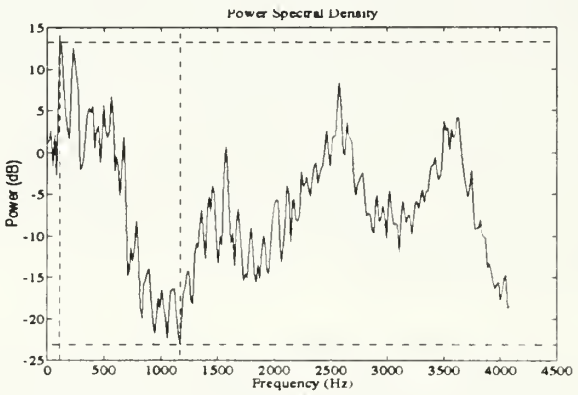
In this tutorial, we measure the largest and smallest magnitude and corresponding frequencies in the file seatsit.voc. Load seatsit.voc according to the instructions given in the vectarma tutorial. The following plot should be displayed after the spectrum analyzer has been started using the command "sanalyzr(seatsit)".



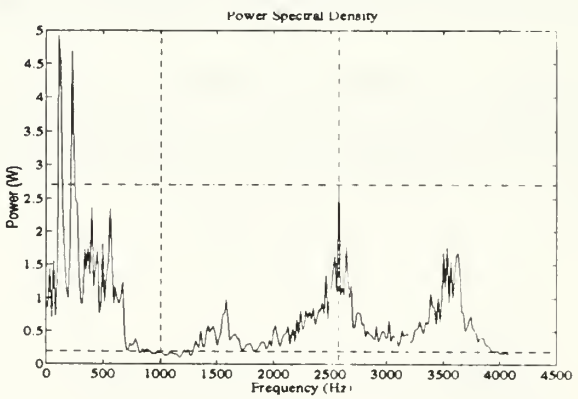
The spectrum shown above uses an enhanced version of the spectrum command from the Matlab Signal Processing Toolbox, named spectrm2.m, allowing user defined windows (vice the default hanning window). The default window in spectrm2 is still a hanning window but the window can be changed after sanalyzr is started by selecting another window from the Window pulldown menu. The Matlab spectrum command uses the Welch method

of spectral estimation and is used here with no window overlap. The default FFT length is 1024 points. Since only the positive frequencies up to half the sampling frequency are displayed. In addition, the DC component is not displayed.

With the mouse, drag the spectrum analyzer cursors to within close proximity of the highest and lowest peaks, as shown below. When the cursors are in the proximity of the peaks, the "<<" and ">>" pushbuttons may be used to align the cursor exactly on the peaks. The frequency and magnitude measurements are displayed in the cursor readouts.



Note a linear scale is also available from the pulldown menu to display the spectrum under study.



Finally, note that the user can generate a variety of baseband and passband communication signals using commands available in the SPC Toolbox and then study their spectrums using sanalyzr.

TUTORIAL

BPSK Modulation/Demodulation

This tutorial focuses on using commands from the SPC Toolbox to generate a BPSK signal containing a known message, simulate transmission of that signal through a noisy channel and then demodulate and recover the transmitted message from the "received" signal. While this tutorial uses BPSK as the modulation method, BFSK or OOK modulation could easily be substituted in its place.

This tutorial will use the following parameters:

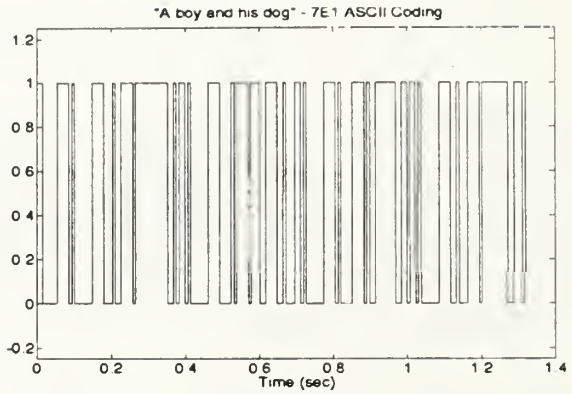
BPSK modulation,
7-bit, even parity ASCII with one stop bit coding,
128 bits-per-second bit rate,
Sampling frequency of 8192 Hz,
Carrier frequency of 1024 Hz,
15 dB in-band, signal-to-noise ratio.

One popular way to transmit binary data is through the use of modems such as those found on most personal computers. A modem transmits data one character at a time. Each character is transmitted using one bit to mark the start of each new character and one or two bits to mark its end. The character itself is transmitted as either seven or eight bits with an optional parity bit as a means of performing some error checking. The function **str2masc** converts a character string to a vector of 1's or 0's representing the binary stream that would be sent if the string was transmitted by a modem.

```
msg = 'A boy and his dog';
m = str2masc(msg,7,'e',1);
m(1:10)
ans =
     1      (start bit)
     1      (data bits)
     0
     0
     0
     0
     0
     1
     1      (parity bit)
     1      (stop bit)
```

The **unipolar** command can be used to generate a baseband rendition of this signal.

```
fs = 8192; Rb = 128;
msgwave = unipolar(Rb,fs,m);
plot((0:length(msgwave)-1)/fs,msgwave);
set(gca,'YLim',[-0.25 1.25]);
title(' "A boy and his dog" - 7E1 ASCII Coding');
xlabel('Time (sec)');
```



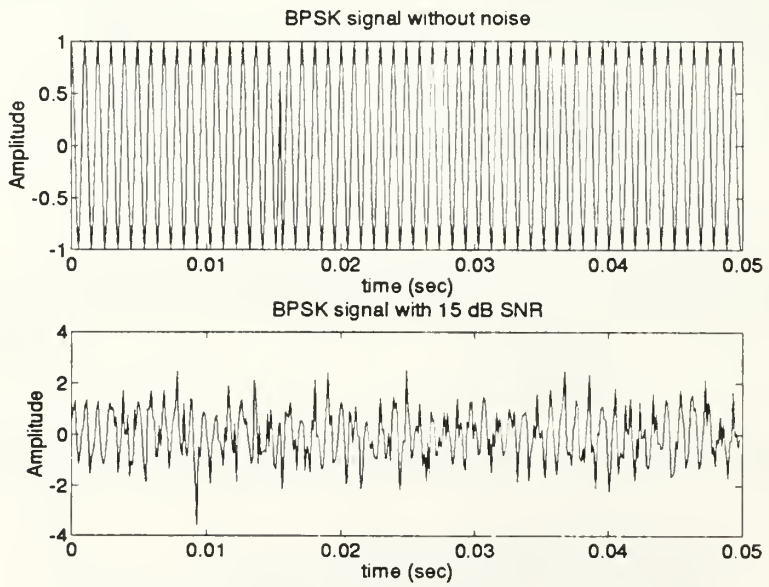
Two commands are available to directly create BPSK signals. The first is the **bpsk** command but this only creates random messages. The second command, **bpskmsg**, is used to create a BPSK signal containing a specific binary message.

```
fc = 1024;
p = bpskmsg(Rb,fc,fs,m);
```

To simulate a noisy transmission channel, the signal is mixed with additive Gaussian white noise such that the in-band signal-to-noise ratio is 15 decibels. The **setsnrbw** command performs this function.

```
SNR = 15;
r = setsnrbw(p,randn(length(p),1),SNR,fc,Rb,fs);
```

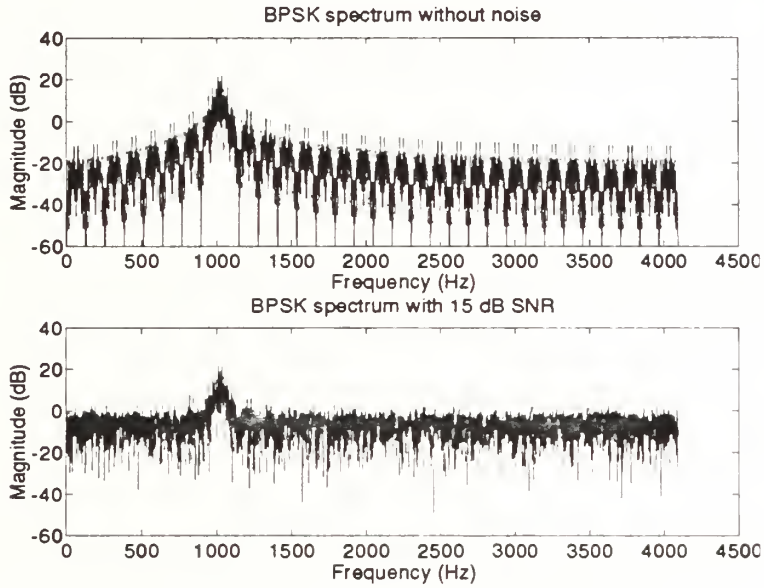
Taking a look at how this signal appears before and after "transmission" is easy to do with the **plottime** command.




```
subplot(2,1,1),plottime(p,0.05),
title('BPSK signal without noise');
```

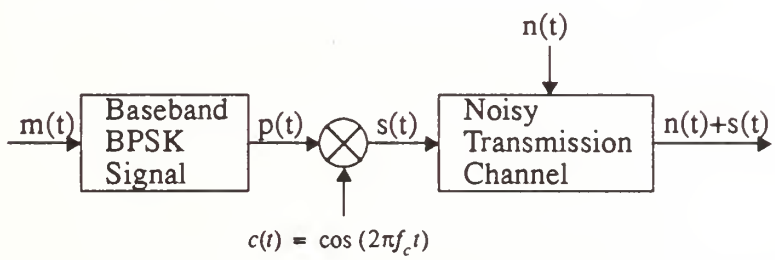
```
xlabel('time (sec)');ylabel('Amplitude');
subplot(2,1,2),plottime(r,0.05);
title(['BPSK signal with ' int2str(SNR) ' dB SNR']);
xlabel('time (sec)');ylabel('Amplitude');
```

Also, plotting the noise-free and noisy power spectral densities (periodograms) is easy with the lperigrm command for a log scaled look (decibels) or wperigrm for a linear scaled look (relative watts).



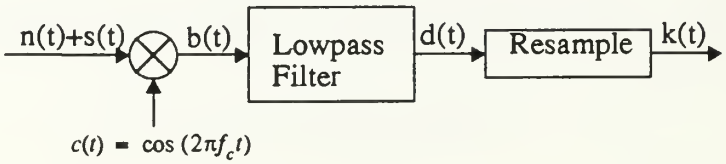

```
subplot(2,1,1),lperigrm(p)
title('BPSK spectrum without noise');
subplot(2,1,2),lperigrm(r)
title(['BPSK spectrum with ' int2str(SNR) ' dB SNR']);
```

Thus far, the following has been simulated:



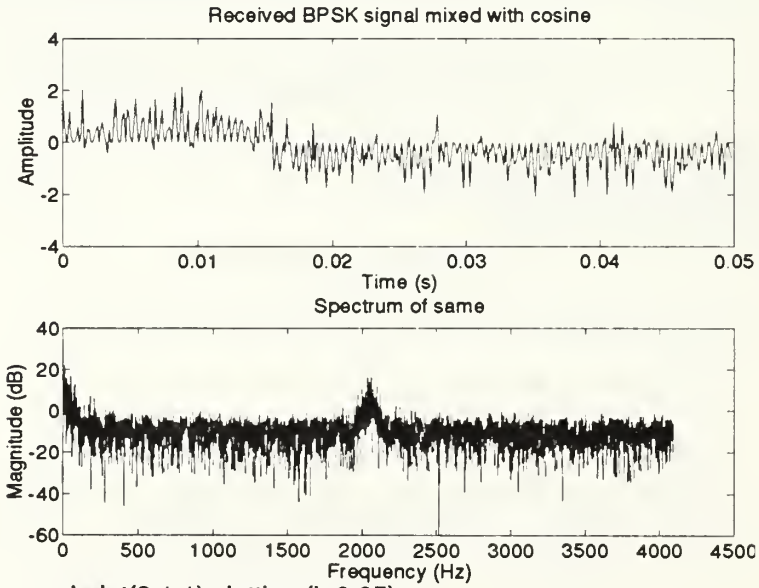
First, the message was encoded in binary using str2masc to produce *m(t)*. Second, *m(t)* was keyed as a baseband BPSK signal, *p(t)*, and modulated onto a carrier, *c(t)*, creating the transmitted signal *s(t)*. The bpskmsg command actually did this operation by converting *m(t)* into a polar signal using the antpodal command and then modulating this signal onto a sinusoidal carrier using double-sideband, suppressed-carrier modulation with the dsbsc command. Third, Gaussian white noise was added to the transmitted signal to simulate a noisy transmission channel using the setsnrbw command.

The block diagram below outlines the system, known as a product detector, which is used to demodulate the "received" signal.



The reader may recall that demodulation of PSK signaling can only be performed using coherent detection. To the uninitiated, this means that the sinusoidal signal shown in the diagram above must be in-phase with the carrier signal. This is easy to do as the bpskmsg command generates a carrier with zero phase shift. As long as the sinusoid above is created with zero phase shift (at the same sampling frequency), the sinusoid and the carrier are in-phase and hence, coherent detection can be performed.

```
t = 0:1/fs:(length(r)-1)/fs; c = cos(2 * pi * fc .* t');
b = c .* r;
```



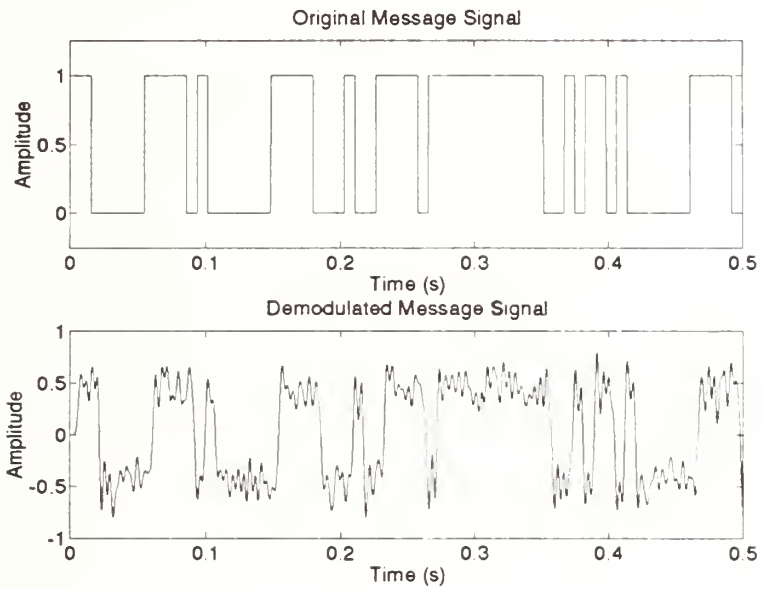

```
subplot(2,1,1),plottime(b,0.05);
title('Received BPSK signal mixed with cosine')
subplot(2,1,2),lperigrm(b)
title('Spectrum of same')
```

The above plot shows that the spectrum of *b(t)* has an image at baseband and at twice the carrier frequency. Also note that the magnitude of the baseband image is greater than the image at twice the carrier frequency.

The next step requires filtering *b(t)* using a lowpass filter. This filtering removes the image

at twice the carrier frequency leaving a baseband polar signal. A number of factors come into play when selecting the type of filter and its cutoff frequency for use in this step. These include the roll-off characteristics of the filter and the number signal side lobes to be included in the passband of the filter. Including more signal side lobes will include more harmonic frequencies from the baseband signal (which is good) but it will also include more noise (which is bad). More harmonics frequencies would make a "squarer" waveform but more noise could offset that benefit. Obviously, trade-offs have to be made. Here, a tenth order, type 1 Chebychev lowpass filter with 1 dB of ripple in the passband is used. The passband will include the main lobe and one side lobe.

```
[bb,aa] = cheby1(10,1,4*Rb/fs);
d = filter(bb,aa,b);
```




```
subplot(2,1,1),plottime(msgwave,0.5);
title('Original Message Signal');
subplot(2,1,2),plottime(d,.5);
title('Demodulated Message Signal');
```

As can be seen, the result indeed resembles a noisy polar waveform. Comparing this plot to the first plot of the binary waveform reveals a phase delay. This delay is caused by the non-causal property of the lowpass IIR filter used in this tutorial.

The final step is to recover the information. To do this, the signal *d(t)* is resampled taking one sample per bit. To properly resample *d(t)*, care must be taken to sample each bit only when it has stabilized after a bit transition but before a transition to the next bit starts. Because of the filter delay, *d(t)*, can simply be sample at times when bit transitions would have occurred (every f_s/R_b samples).

```
k = d(fs/Rb:fs/Rb:length(d));          % fs/Rb = samples-per-bit
```

Once resampled, *k(t)* still represents an analog "voltage." An analog system would use a

threshold detector at this point to convert the analog signal to a binary signal. A logical comparison operator in Matlab can be used to simulate a threshold detector. In this case, the optimal threshold voltage is zero and everytime the signal *k(t)* is greater than this value, it is considered a binary 1.

```
k = k > 0;              % convert to binary
```

This results in a binary representation of the received message. This binary representation can be compared to the binary representation of the original message to compute the number of bit errors.

```
biterror = sum(k ~= m)
biterror =
        0
```

Last, the masc2str command can be used to convert the received binary signal into a readable form.

```
masc2str(k,7,'e',1)
ans =
        A boy and his dog
```

For further study:

1. Change various parameters in the above simulation and observe the results. Candidate parameters include:
   - Signal-to-noise ratio.
   - Modulation method.
   - Lowpass filter type, cutoff frequency, attenuation, ripple, etc.

2. Add interfering signals (i.e. voice, BFSK, etc.) in addition to the Gaussian white noise and attempt to recover the modulated information. This environment can be generated by first creating an environment containing the primary signal with Gaussian white noise (as in this tutorial), and then generating a second environment by creating the interfering signal and using the first environment as the noise input to the setsnrbw command.

3. Pad the beginning and end of the signal with arbitrary amounts of white noise (scaled appropriately with the σ scaling factor returned by the setsnrbw command). Write a command to "synch-up" with the signal in order to use coherent demodulation.

4. Experiment using error correcting codes and low SNR's.

5. Instead of subsampling the demodulated signal, try using an integrator or some other bit detection scheme.

ANTPODAL

Purpose

Generate a baseband, antipodal [-1,1] signal.

Synopsis

```
y = antpodal(fb,msg);
y = antpodal(fb,msg,fs);
```

Description

antpodal(fb,msg) - Generates a baseband antipodal signal at bit rate, *fb*. Sampling frequency defaults to 8192 Hz. If *msg* is a scalar, a message is generated as a random sequence of length *msg* with Pr(-1)=Pr(1)=0.5. If *msg* is a vector, it must be a vector of 0's and 1's. The length of the output vector, *y*, can be computed by the formula:

$$length = nbrbaud \cdot floor\left(\frac{f_s}{f_b}\right)$$

antpodal(fb,msg,fs) - Sampling frequency is set to *fs*.

Note: If the bit rate, *fb*, and the sampling frequency, *fs*, are both equal to one, the resulting vector will contain alternating values of [-1,1].

Example

Generate 30 bits of a 50 bit-per-second signal sampled at 8000 Hz. Use this baseband signal as the modulating signal input to dsbsc to generate phase-shift keyed signal.

```
x = antpodal(50,30,8000);
y = dsbsc(x,2000,8000);
```

Limitations

- The probability 1 occurs will not be exact for a small number of bits.
- If the sampling frequency is not an exact multiple of the bit rate, unexpected results may occur. For example: antpodal(9,18,20) will produce an output of 36 samples vice the expected 40 samples (i.e. 18 / 9 = 2 seconds at 20 samples-per-second). This occurs since *floor(fs/fb)* = 2 samples-per-bit.

See Also

unipolar, sqwave

AR_BURG

Purpose

Compute parameters for an AR model using the Burg method.

Synopsis

[a,b0,S,gamma,sigma] = ar_burg(x,P)

Description

ar_burg(x,P) - Computes the coefficients of an order P Auto-Regressive Model (AR) of the data given in vector x via the Burg method as put forth in [1]. The output is specified by:

a = filter coefficients
b_0 = gain
S = prediction error variance

The gain term, b_0, is not computed in the Burg algorithm itself. It is computed using "brute force" by generating the impulse response of the model and then applying:

$$gain = sqrt\left(\frac{P_x}{P_{model}}\right)$$

Example

Estimate the parameters for a second order AR model for the data sequence {1, -2, 3, -4, 5}. (Example 9.6 in [1].)

```
[a,b0,S] = ar_burg([1 -2 3 -4 5],2);
a =        1.0000
           2.0000
           1.0000
b0 =       1.0000
S =     2.8422e-014
```

Algorithm

$$a_P = \begin{bmatrix} a_{P-1} \\ \emptyset \end{bmatrix} - \gamma_P^* \begin{bmatrix} \emptyset \\ \overleftarrow{a}_{P-1}^* \end{bmatrix} \qquad \sigma^2_{\epsilon_P} = (1 - |\gamma_P|^2)\,\sigma^2_{\epsilon_{P-1}}$$

See Also

ar_corr, ar_covar, ar_mdcov, ar_svd, ar_prony, ar_durbn, ar_shank

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 545-548, Prentice-Hall, 1992.

AR_CORR

Purpose

Compute parameters for an AR model using the autocorrelation method.

Synopsis

```
[a,b0,S,s] = ar_corr(x,P)
```

Description

ar_corr(x,P) - Computes the coefficients of an order P Auto-Regressive Model (AR) of the data given in vector x via the autocorrelation method as put forth in [1]. The output is specified by:

a = filter coefficients
b0 = gain
S = minimum sum of squares
s = estimate for prediction error variance

Example

Estimate the parameters for a second order AR model for the data sequence {1, -2, 3, -4, 5}. (Example 9.6 in [1].)

```
[a,S,s] = ar_corr([1 -2 3 -4 5],2);
a =
        1.0000
        0.8140
        0.1193
S =
       25.5404
s =
        3.6486
```

Algorithm

$$b_0 = \sqrt{S} \qquad X^H X a = \begin{bmatrix} S \\ \emptyset \end{bmatrix}$$

See Also

ar_covar, ar_mdcov, ar_burg, ar_svd, ar_prony, ar_durbn, ar_shank

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 535-541, Prentice-Hall, 1992.

AR_COVAR

Purpose

Compute parameters for an AR model using the covariance method.

Synopsis

[a,b0,S,s] = ar_covar(x,P)

Description

ar_covar(x,P) - Computes the coefficients of an order P Auto-Regressive Model (AR) of the data given in vector x via the covariance method as put forth in [1]. The output is specified by:

a = filter coefficients
b_0 = gain
S = minimum sum of squares
s = estimate for prediction error variance

The gain term, b_0, is not computed in the covariance algorithm itself. It is computed using brute force by generating the impulse response of the model and then applying:

$$gain = sqrt\left(\frac{P_x}{P_{model}}\right)$$

Example

Estimate the parameters for a second order AR model for the data sequence {1, -2, 3, -4, 5}. (Example 9.6 in [1].)

```
[a,b0,S,s] = ar_covar([1 -2 3 -4 5],2);
a =      1.0000
         2.0000
         1.0000
b0 =     1.0000
S =   2.8422e-013
s =   5.6843e-014
```

Algorithm

$$(X^{*T} X)\, a = \begin{bmatrix} S \\ \emptyset \end{bmatrix}$$

See Also

ar_corr, ar_mdcov, ar_burg, ar_svd, ar_prony, ar_durbn, ar_shank

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 535-541, Prentice-Hall, 1992.

AR_DURBN

Purpose

Compute parameters for an ARMA model using the Durbin method.

Synopsis

```
[a,b] = ar_durbn(x,P,Q,L)
```

Description

ar_durbn(x,P,Q,L) - Computes the coefficients of an order *P* Auto-Regressive, order *Q* Moving Average Model of the data given in vector *x*. The *L* parameter can be used to set the order of the intermediate AR model used in the algorithm. If not provided, *L* defaults to *5 * Q* or the length of *x minus 1*, whichever is smaller. The intermediate AR model is first found using the covariance method. This model is them used to generate new data from which the MA parameters are found using the Durbin method. The AR parameters are found using the covariance method. The output arguments are:

a = AR parameters
b = MA parameters

Limitations

The Durbin method is not well suited for small data sets.

See Also

ar_corr, ar_covar, ar_mdcov, ar_burg, ar_svd, ar_prony, ar_shank

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 558-560, Prentice-Hall, 1992.

AR_LEVIN

Purpose

Compute parameters for an AR model using the Levinson recursion.

Synopsis

```
[a,s]=ar_levin(r,P)
```

Description

ar_levin(r,P) - Computes the coefficients of an order *P* Auto-Regressive Model from values of the autocorrelation vector *r = [R(0) R(1) R(2) ... R(P-1)]*. The autocorrelation must be symmetric (*R(1) = R*(-1), R(2) = R*(-2)*, etc,.). The output arguments are:

a = AR parameters
s = linear prediction error variance

Example

Given r = [3 2 1 0 0 0 ...], find the second order AR model. (Example 8.1 in [1])

```
[a,s] = ar_levin([3 2 1 0],2);
a =      1.0000
        -0.8000
         0.2000
s =      1.6000
```

Algorithm

Initialization:

$$r_0 = R_x[1] \qquad a_0 = 1 \qquad \sigma^2 = R_x[0]$$

Recursion:

$$\Delta_p = \mathbf{r}_{p-1}^{*T}\ \tilde{\mathbf{a}}_{p-1}$$

$$\gamma_p = \frac{\Delta_p}{\sigma_{p-1}^{2}}$$

$$\mathbf{a}_p = \begin{bmatrix} \mathbf{a}_{p-1} \\ \emptyset \end{bmatrix} - \gamma_p \begin{bmatrix} \emptyset \\ \tilde{\mathbf{a}}_{p-1}^{*} \end{bmatrix}$$

$$\sigma_p = \sigma_{p-1}^2 - \gamma_p \Delta_p^*$$

See Also

ar_corr, ar_covar, ar_mdcov, ar_burg, ar_prony, ar_durbn, ar_shank

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 422-430, Prentice-Hall, 1992.

AR_MDCOV

Purpose

Compute parameters for an AR model using the modified covariance method.

Synopsis

```
[a,b0,S,s] = ar_mdcov(x,P)
```

Description

ar_mdcov(x,P) - Computes the coefficients of an order P Auto-Regressive Model (AR) of the data given in vector x via the modified covariance method as put forth in [1]. The output is specified by:

a = filter coefficients
b_0 = gain
S = minimum sum of squares
s = estimate for prediction error variance

The gain term, b_0, is not computed in the modified covariance algorithm itself. It is computed using brute force by generating the impulse response of the model and then applying:

$$gain = sqrt\left(\frac{P_x}{P_{model}}\right)$$

Example

Estimate the parameters for a second order AR model for the data sequence {1, -2, 3, -4, 5}. (Example 9.6 in [1].)

```
[a,b0,S,s] = ar_mdcov([1 -2 3 -4 5],2);
a =      1.0000
         2.0000
         1.0000
b0 =     1.0000
S =   2.8422e-014
s =   5.6843e-015
```

Algorithm

$$(X^{*T}X + \tilde{X}^T\tilde{X}^*)a = \begin{bmatrix} S \\ \emptyset \end{bmatrix}$$

See Also

ar_corr, ar_covar, ar_burg, ar_svd, ar_prony, ar_durbn, ar_shank

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 542-544, Prentice-Hall, 1992.

AR_PRONY

Purpose

Compute parameters for an ARMA model using the Prony method.

Synopsis

[a,b] = ar_prony(x,P,Q)

Description

ar_prony(x,P,Q) - Computes the coefficients of an order P Auto-Regressive, order Q Moving Average Model of the data given in vector x. The AR parameters are found using the covariance method. MA parameters are found using the Prony method. The output arguments are:

a = AR parameters
b = MA parameters

See Also

ar_corr, ar_covar, ar_mdcov, ar_burg, ar_svd, ar_durbn, ar_shank

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 550-555, Prentice-Hall, 1992.

AR_SHANK

Purpose

Compute parameters for an ARMA model using the Shank method.

Synopsis

```
[a,b] = ar_shank(x,P,Q)
```

Description

ar_shank(x,P,Q) - Computes the coefficients of an order *P* Auto-Regressive, order *Q* Moving Average Model of the data given in vector *x*. The AR parameters are found using the covariance method. MA parameters are found using the Shank method. The output arguments are:

a = AR parameters
b = MA parameters

See Also

ar_corr, ar_covar, ar_mdcov, ar_burg, ar_svd, ar_prony, ar_durbn

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 555-558, Prentice-Hall, 1992.

AVSMOOTH,MDSMOOTH

Purpose

Smooth a curve using an average or median smoothing filter.

Synopsis

```
y = avsmooth(x,L);
y = mdsmooth(x,L);
```

Description

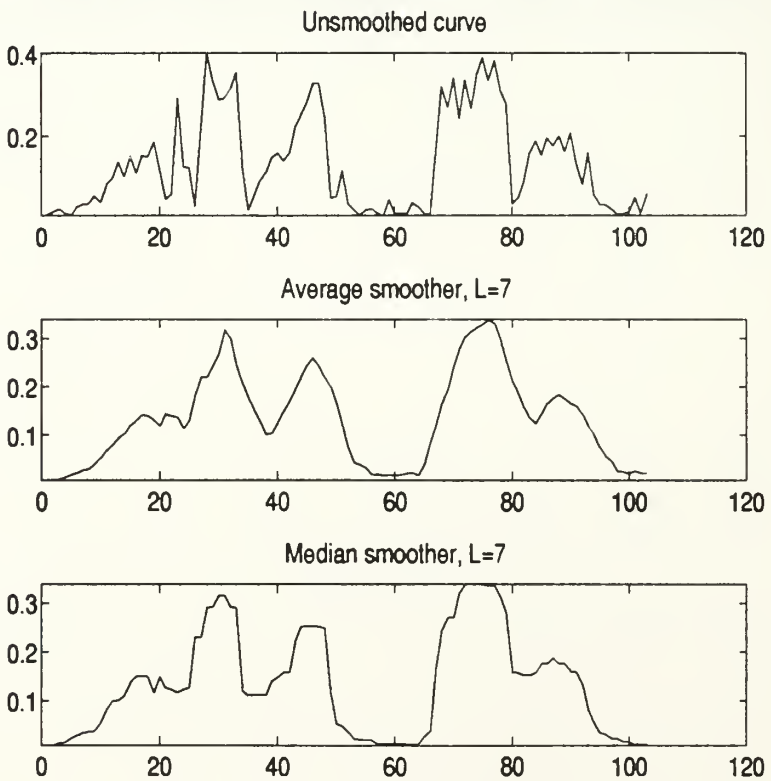
avsmooth(x,L) - Smooths the curve specified in vector *x* using an average smoothing filter of length *L*.

mdsmooth(x,L) - Smooths the curve specified in vector *x* using an median smoothing filter of length *L*.

Example

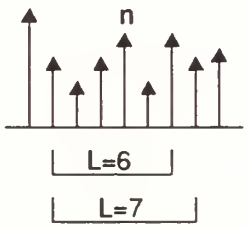
Compute the short-time energy of the speech signal specified in *x* and enhance the resulting curve using an average smoother of length 7.

```
y = sp_steng(x,64,30,'hamming');
yy = avsmooth(y,7);
yyy = mdsmooth(y,7);
```



Algorithm

A non-causal smoothing algorithm is used. If L is odd, $\frac{L-1}{2}$ samples before and after the current sample are used in computing the average. If L is even, $L/2$ samples before and $\frac{L}{2}-1$ samples after the current sample are used in computing the average. It is recommended an odd number for L be used to prevent any biasing an even number for L can cause.



Limitations

Zero padding is used on either end of the input vector and then the vector is trimmed to produce a non-causal filter. Therefore, samples within $L/2$ samples from either end will be subject to a transient which distorts the curve toward zero. This distortion becomes more exaggerated the further the absolute value of the samples in this region are from zero.

See Also

mdsmooth

BFSK, BFSKMSG

Purpose

Generate binary frequency-shift keyed, bandpass signals.

Synopsis

```
y = bfsk(Rb,shift,fc);
y = bfsk(Rb,shift,fc,duration);
y = bfsk(Rb,shift,fc,duration,fs);
y = bfskmsg(Rb,shift,fc,msg);
y = bfskmsg(Rb,shift,fc,fs,msg);
```

Description

bfsk(Rb,shift,fc) - Generates one second of a binary frequency-shift keyed signal with a bit rate of *Rb* bits-per-second and a frequency shift of *shift* centered at a carrier frequency, *fc*. Default sampling rate is 8192 Hz.

bfsk(Rb,shift,fc,duration) - Generates *duration* seconds of the binary frequency-shift keyed signal.

bfsk(Rb,shift,fc,duration,fs) - Set the sampling frequency to *fs*.

bfskmsg(Rb,shift,fc,msg) or bfskmsg(Rb,shift,fc,fs,msg) - Generate a bandpass binary frequency-shift keyed signal with a bit rate of *Rb* bits-per-second, a frequency shift, *shift*, centered at a carrier frequency, *fc*, sampled at sampling frequency, *fs*, and containing the message, *msg*. If *fs* is not given, *fs* defaults to 8192 Hz.

Example

Generate a 64 bit-per-second, binary frequency-shift keyed signal at a carrier frequency of 2048 Hz and a frequency shift of 128 Hz lasting one-and-one-half seconds. Play the signal over the workstation's speaker.

```
y = bfsk(64,128,2048,1.5);
sound(y,8192);
```

Algorithm

The binary frequency-shift keyed signal is generated by first creating two on-off keyed signals at the desired bit/sampling rates and combining them. The on-off keyed signals are generated such that the mark signal is generated at *fc+shift/2* and the space signal is generated at *fc-shift/2*. After the signals are generated, they are added to form a BFSK signal. Note: if *shift* is an integer multiple of the bit rate (*shift=n*Rb* where *n*=1,2,...), the resulting BFSK signal is coherent (i.e. has continuous phase).

Limitations

See limitations under unipolar.

See Also

unipolar, dsbsc, ook, ookmsg, bpsk, bpskmsg

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 332-344, 351-356, Macmillan, 1990.

BPSK, BPSKMSG

Purpose

Generate binary phase-shift keyed, bandpass signals.

Synopsis

```
y = bpsk(Rb,fc);
y = bpsk(Rb,fc,duration);
y = bpsk(Rb,fc,duration,fs);
y = bpsk(Rb,fc,msg);
y = bpskmsg(Rb,fc,fs,msg);
```

Description

bpsk(Rb,fc) - Generates one second of a binary phase-shift keyed signal with a bit rate of *Rb* bits-per-second centered at a carrier frequency, *fc*. Default sampling rate is 8192 Hz.

bpsk(Rb,fc,duration) - Generates *duration* seconds of a phase-shift keyed signal.

bpsk(Rb,fc,duration,fs) - Sets the sampling frequency to *fs*.

bpskmsg(Rb,fc,msg) or bpskmsg(Rb,fc,fs,msg) - Generates a binary phase-shift keyed signal with a bit rate of *Rb* bits-per-second, at a carrier frequency, *fc*, sampled at sampling frequency, *fs*, and containing the message, *msg*. If *fs* is not given, *fs* defaults to 8192 Hz.

Example

Generate a 64 bit-per-second, binary phase-shift keyed signal at a carrier frequency of 2048 Hz lasting one-and-one-half seconds. Play the signal over the workstation's speaker.

```
y = bpsk(64,2048,1.5);
sound(y,8192);
```

Algorithm

The binary phase-shift keyed signal is generated by first creating a binary antipodal signal at the desired bit/sampling rates and then applying a double-sideband, suppressed-carrier modulation scheme to the resulting baseband signal at the desired carrier frequency.

Limitations

See limitations under antpodal.

See Also

bpskmsg, ook, ookmsg, bfsk, bfskmsg

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 332-336, Macmillan, 1990.

COMMON CONTROLS

This toolbox contains a number of interactive "dodads" driven by popupmenus, pushbuttons and other graphical controls. Several of these controls are common to a number of dodads and are discussed below. For specific instructions on the use of Matlab popupmenus, pushbuttons, edit boxes, and sliders, refer to discussions of "Handle Graphics" in the *Matlab User's Manual* and the "uicontrol" statement in the *Matlab Reference Manual* for your system.

Mark Start/Mark End Pushbuttons

Purpose

- Marks the beginning and end of a vector section to be cut, copied or otherwise acted upon.
- Marks cutoff frequencies for filters (lower cutoff = Mark Start, upper cutoff = Mark End).

Use

1. Select desired mark pushbutton (Start or End).
2. After selection, the cursor changes to a crosshair and the plot title changes to either "Mark start with cursor" or "Mark end with cursor" as appropriate.
3. Move the cursor to the desired mark location and click the left mouse button. When placing the mark, only the location on the horizontal (time or frequency) axis is of importance. Placing a mark past either end of the plot axis sets the mark to the first or last value displayed on the horizontal axis.

Result

After placing a mark, the cursor changes back into an arrow and the plot title is cleared. The start mark is displayed as a vertical dashed line and the end mark is displayed as a vertical dash-dot line.

Reset Marks Pushbutton

Purpose

Reset marks to the extremes of the current display. On a "Zoom Full" display, Reset sets the marks to the beginning and end of the vector. On a "Zoom Marked" display, Reset sets the marks to the beginning and end of the zoomed section of the vector.

Use

1. Click the left mouse button while the cursor is over the Reset Marks control.

Result

The respective marks are changed to the first or last value displayed on the axis. When marks are at the axis extremes, they may or may not be visible depending upon whether or not the plot border obscures them. (This is a problem in version 4.0a of Matlab for Sun workstations that has been fixed in version 4.1.)

Play Popupmenu

Purpose

Send either the entire vector or marked section thereof to the workstation's audio output.

Use

1. If desired, mark a portion of the vector using Mark Start and Mark End.
2. Place the cursor on the Play popupmenu and click the left mouse button once to open the menu.
3. Place the cursor on the desired function (Play Full or Play Marked) and click the left mouse button.

Result

Selecting Play Full outputs the entire vector to the audio output. Selecting Play Marked outputs the portion of the vector between the marks to the audio output. If the marks have not been set, the entire vector is played.

Notes

- On a Sun workstation, the output volume and device (speaker or headphone jack) are set using the Sun operating system's AudioTool. The output sampling frequency is fixed at 8,192 Hz.
- On a PC under Microsoft Windows, the output device is to the soundcard, if one is installed and supported by Matlab. Tools which accompany the soundcard are used to set the output volume. The output sampling frequency is set by the sampling frequency popupmenu.

Zoom Popupmenu

Purpose

- Zooms the display in on a marked section of the vector.
- Zooms the display out to show the full vector.

Use

1. If desired, mark a portion of the vector using Mark Start and Mark End.
2. Place the cursor on the Zoom popupmenu and click the left mouse button once to open the menu.
3. Place the cursor on the desired function (Zoom Full or Zoom Marked) and click the left mouse button.

Result

The requested action is performed. Zoom Marked can be repeatedly used.

Restore Pushbutton

Purpose

Restores the original vector the dodad was called with if a vector argument was supplied. If the dodad was called without an argument, the common vector at the time the dodad was called is restored (see Common Pushbutton).

Use

1. Place the cursor on the Restore pushbutton and click the left mouse button.

Result

The original vector is restored. All prior edits are lost unless previously saved to the Matlab workspace (see Save pushbutton).

Common Pushbutton

Purpose

A "common" vector is maintained between dodads to facilitate loading the current vector from one dodad into others without having to go through a save, quit, and

restart sequence. This common vector is updated after any operation that modifies a vector. For example, using cut and paste in the Vector Edit Dodad or filtering a vector using the Vector Filter Dodad are examples of operations that change a vector. Suppose a vector is filtered using the vector filter dodad and the vector edit dodad is already open. After the filtering is complete, selecting the Common pushbutton loads the filtered signal into the vector edit dodad.

Use

1. Perform an operation that changes a vector in one of the dodads.
2. Place the cursor on the Common pushbutton and click the left mouse button.

Result

A copy of the current common vector is loaded into the dodad.

Save Pushbutton

Purpose

Saves the current vector to the Matlab workspace under a new variable name. This is the only way to save any changes made to a vector.

Use

1. Place the cursor on the Save Pushbutton and click the left mouse button.
2. The Save and Load pushbuttons are replaced by an edit box. If save has already been used, the previously entered vector name are contained in the edit box.
3. Place the cursor on the edit box and click the left mouse button. Using the keyboard, backspace over any previous entry and then type in the desired vector name (following standard Matlab variable name conventions).
4. Press the return key.

Result

The vector is saved to the Matlab workspace.

Load Pushbutton

Purpose

Loads a vector from the Matlab workspace into the dodad.

Use

1. Place the cursor on the Load Pushbutton and click the left mouse button.
2. The Save and Load pushbuttons are replaced by an edit box. If load has already been used, the previously entered vector nameis contained in the edit box.
3. Place the cursor on the edit box and click the left mouse button. Using the keyboard, backspace over any previous entry and then type in the desired vector name (following standard Matlab variable name conventions).
4. Press the return key.

Result

The vector is loaded from the Matlab workspace.

Notes

The vector has to exist in the Matlab workspace. If the vector is not in the Matlab workspace or the variable name is misspelled, the error message "Invalid variable name or vector not in workspace!" is printed in the Matlab command window and the Save and Load pushbuttons are restored. The Load pushbutton needs to be reselected to retry.

Close Pushbutton

Purpose

Closes the dodad and any child graphic windows that may be open.

Use

1. Place the cursor on the Close Pushbutton and click the left mouse button.

Result

Any child graphic windows are closed along with the dodad itself. The vector is NOT saved upon exit. To save the vector before exiting, see Save pushbutton.

Snapshot Pushbutton

Purpose

Opens a new figure window and recreates the current dodad plot, effectively taking a "snapshot." This new figure window is then forgotten by the dodad and remains open after the dodad is closed. Once the new figure window has been created, the figure window can be treated like any other Matlab figure window and either closed, edited or printed. This allows the user to take a snapshot of the current dodad plot, add/change its title and then print the image to a postscript file or printer, as desired.

Use

1. Place the cursor on the Snapshot pushbutton and click the left mouse button.

Result

The current plot is recreated in a new figure window as discussed above. Before printing or otherwise handling this new figure, it should be made the current figure with the use of the command figure(*h*) here *h* is the figure window number. It is up to the user to close this figure window. The number of snapshots displayed at one time is only limited by the workstations memory.

Sampling Frequency Popupmenu

Purpose

Set the sampling frequency.

Use

1. Place the cursor on the Sampling Frequency Popupmenu and click the left mouse button once to open the menu.
2. Place the cursor on the sampling frequency and click the left mouse button. The popupmenu contains sampling frequencies of 1, 4096, 8000, 8192, 11024, 22048 and 44096 Hz. If the sampling frequency is not one of those mentioned earlier, the desired value can be entered by selecting the "User" menu item. In this case, an edit box appears in place of the popupmenu. Enter the desired sampling frequency into this edit box and press return.

Result

The display is redrawn with a new time axis according to the sampling frequency.

Notes

In the popupmenu, the "User" menu item is replaced by the sampling frequency if it is entered using the edit box. To subsequently change the sampling frequency after using the edit box, the last sampling frequency entered has to be selected. This is always the last menu item on the popupmenu. If a sampling frequency of 1

Hz is selected, the horizontal shows the vector indices.

Time/Spectrum Radiobuttons

Purpose

When provided, changes the vector display between the time and frequency domains.

Use

1. Place the cursor on the desired radiobutton option and click the left mouse button.

Result

The display is redrawn appropriately.

Notes

- If the display is zoomed before switching domains, the display is still zoomed after switching back.
- The spectrum displayed is generated by the Signal Processing Toolbox spectrum command using 1024 points (512 points displayed).

COSWAVE,SINWAVE

Purpose

Generate a baseband, cosine or sine wave.

Synopsis

```
y = coswave(fb);
y = coswave(fb,duration);
y = coswave(fb,duration,fs);
y = coswave(fb,duration,fs,phaseshift);
y = sinwave(fb);
y = sinwave(fb,duration);
y = sinwave(fb,duration,fs);
y = sinwave(fb,duration,fs,phaseshift);
```

Description

coswave(fb) or sinwave(fb) - Generates one second of a sinusoidal wave at cycle frequency, *fb*, and a sampling frequency of 8192 Hz.

coswave(fb,duration) or sinwave(fb,duration) - Generates *duration* seconds of a sinusoidal wave at cycle frequency, *fb*, and a sampling frequency of 8192 Hz.

coswave(fb,duration,fs) or sinwave(fb,duration,fs) - Generates *duration* seconds of a sinusoidal wave at cycle frequency, *fb*, and sampling frequency, *fs*.

coswave(fb,duration,fs,phaseshift) or sinwave(fb,duration,fs,phaseshift) - Generates *duration* seconds of a sinusoidal wave at cycle frequency, *fb*, and sampling frequency, *fs*, with phase shift, *phaseshift*.

Examples

Use a 10 Hz sinusoidal modulating signal to generate an AM signal with a modulation index of 0.5 and a carrier frequency of 50 Hz, sampled at 128 Hz.

```
y = coswave(10,128,60);
yy = dsblc(y,0.5, 50, 128);
```

Generate 3425 samples of a 800 cycle-per-second sinusoidal wave sampled at 7000 Hz.

```
y = coswave(800,3425/7000,7000)
```

See Also

sqwave, triwave, sawwave

DSBLC

Purpose

Generate a double-sideband, large-carrier amplitude modulated signal.

Synopsis

```
y = dsblc(modsig,m,fc,fs)
```

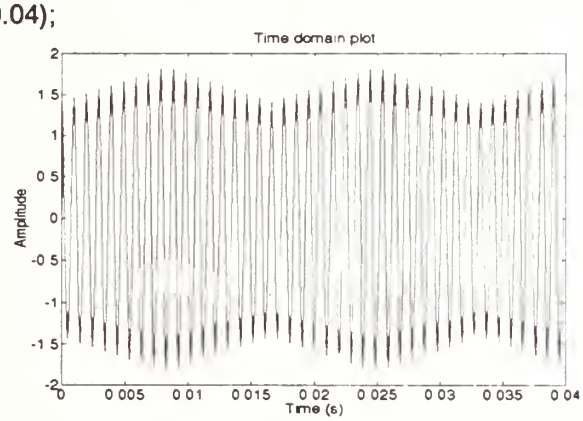
Description

dsblc(modsig,m,fc,fs) - Generates a double-sideband, large-carrier amplitude modulated signal, *y*. The signal contained in the vector *modsig* is modulated onto a sinusoidal carrier of amplitude 1 at carrier frequency, *fc*. The maximum deviation of the modulating signal must vary evenly about zero [i.e. **(max(modsig) == abs(min(-modsig))**]. *modsig* is multiplied by the modulation index, *m*, before being modulated onto the carrier. The length of the signal returned is equal to the length of *modsig*. The carrier is generated as a cosine with zero phase shift. The sampling frequency, *fs*, must be the same sampling frequency as that used to generate the modulating signal.

Example

Modulate one second of a 60 cycle triangular wave using a carrier of 1024 Hz with a modulation index of 0.3.

```
x = triwave(60,30);
y = dsblc(x,0.3,1024);
plottime(y,0.04);
```



Algorithm

The standard equation for double-sideband, suppressed-carrier amplitude modulation with zero phase shift.

$$y(t) = (1 + m\,f(t)) \cos(2\pi f_c t)$$

See Also

dsbsc, modindex, envelope

DSBSC

Purpose

Generate a double-sideband, suppressed-carrier amplitude modulated signal.

Synopsis

```
y = dsbsc(modsig,fc,fs)
```

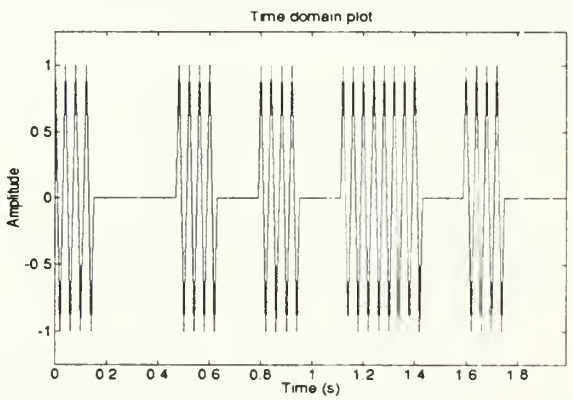
Description

dsbsc(modsig,fc,fs) - Generates a double-sideband, suppressed-carrier amplitude modulated signal, *y*. The signal contained in the vector *modsig* is modulated onto a sinusoidal carrier of amplitude 1 with a carrier frequency *fc*. The length of the modulated signal returned is equal to the length of the *modsig* vector. The carrier is generated as a cosine wave with zero phase shift. The sampling frequency, *fs*, must be the same sampling frequency used to generate the modulating signal.

Example

Generate 16 bits of an 8 bit-per-second signal sampled at 128 Hz. Use this baseband signal as the modulating signal input to dsbsc to generate an on-off keyed signal.

```
x = unipolar(8,[1 0 0 1 0 1 0 1 1 0 0 1 1 0 1],128);
y = dsbsc(x,32,128);
```



Algorithm

The standard equation for double-sideband, suppressed-carrier amplitude modulation with zero phase shift is given by:

$$y(t) = m(t)\cos(2\pi f_c t)$$

See Also

dsbsc

ENVELOPE

Purpose

Demodulate an amplitude modulated signal using envelope (noncoherent) detection.

Synopsis

[y,m] = envelope(x);

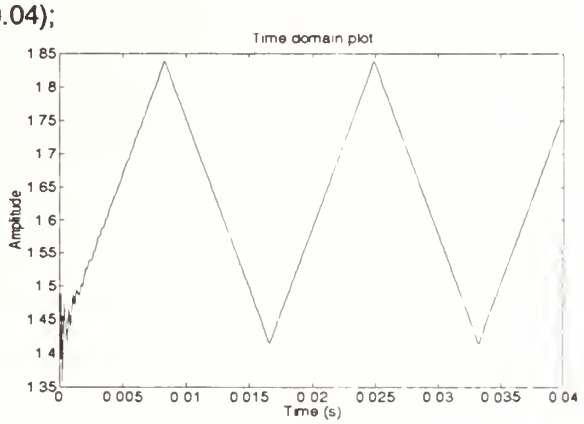
Description

envelope(X) - Returns the AM envelope of the input signal, x, and the modulation index, m.

Example

Demodulate using envelope detection the DSBLC modulated triangular wave used in the DSBLC command example.

```
x = triwave(60,30);
y = dsblc(x,0.3,1024)
z = envelope(y);
plottime(z,0.04);
```



(Note the transients due to the FIR implementation of the Hilbert Transform.)

Algorithm

The complex envelope of the signal x is obtained by computing the Hilbert transform of x. The resulting real envelope is the magnitude of the complex envelope.

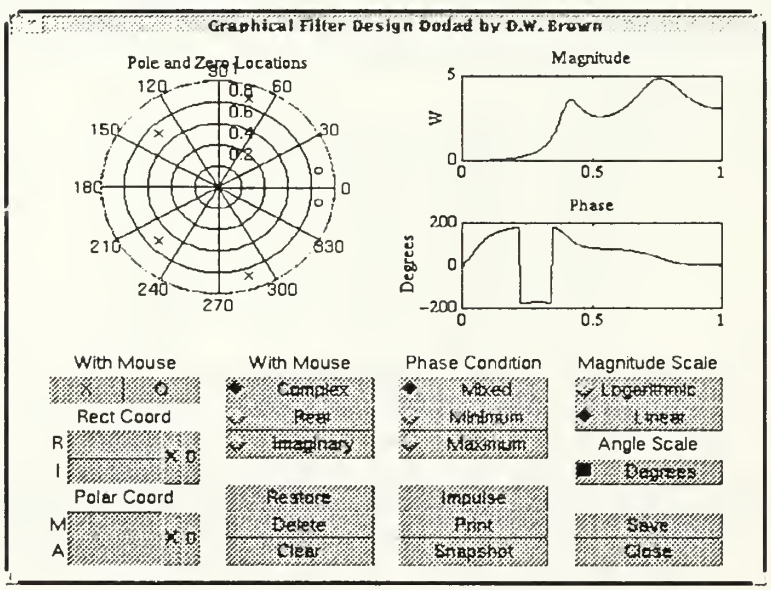
See Also

dsbsc, dsblc, modindex

GRAFILTR

The Graphical Filter Design Dodad provides an interactive, graphical environment within Matlab to design digital filters through individual (complex-conjugate pair) placement of poles and zeros on the pole-zero plot of the filter transfer function. With the Graphical Filter Design Dodad, you can:

- Place complex, real or purely imaginary poles and zeros onto the pole-zero plot.
- Place complex, real or purely imaginary poles and zeros onto the pole-zero plot by specifying the real and imaginary parts.
- Place complex, real or purely imaginary poles and zeros onto the pole-zero plot by specifying the magnitude and angle (degrees or radians).
- Move pole and zero locations using the mouse.
- Delete poles or zeros from the design using the mouse.
- Force the filter transfer function to minimum or maximum phase condition.
- View the filter frequency and phase response as the design progresses.
- Generate the impulse response of the filter.
- Print the filter parameters.
- Save the filter transfer function coefficients.



Starting the Graphical Filter Design Dodad

The Graphical Filter Design Dodad can be started in one of two different ways. The first one is to start the **grafiltr** with two vectors as input arguments. In this usage ("**grafiltr(b,a)**"), the arguments *a* and *b* specify the coefficients of a pre-existing transfer function following Matlab filter transfer function conventions. The arguments *a* and *b* are the same as those return by any of the Matlab Signal Processing filter design programs such as **butter** or **cheby1**. See the discussion for the Matlab fil-

ter command for a complete description of the transfer function specification. Once a pre-existing transfer function has been loaded into grafiltr, poles and zeros can be deleted, added and/or moved about in order to "fine tune" or otherwise manipulate the filter response.

The second way to start the filter dodad is to execute the "grafiltr" command without any arguments. This effectively starts grafiltr with a transfer function of 1 (no poles or zeros). Filter design then start from a "clean slate' by adding poles and zeros.

No output arguments are supported. Completed filter designs must be saved using the Save pushbutton as described below.

Note: Due to a unique usage of the Matlab polar command, this program can only be used with Matlab v.4.1 on Sun workstations.

Graphical Filter Design Dodad Specific Controls

In addition to controls discussed under *common controls*, the following controls are specific to the Graphical Filter Design Dodad (grafiltr).

"With Mouse" X and O Pushbuttons

Purpose

Place a pole or zero on the pole-zero plot.

Use

1. Set the property of the pole or zero using the "With Mouse" property radiobuttons describe below.
2. Select the pole, "X", or zero, "O", pushbutton. After selection, the cursor turns into a crosshair and the x-axis label of the pole-zero plot changes to "Mark pole location with cursor" or "Mark zero location with cursor" as appropriate.
3. Move the crosshair to the desired pole or zero location on the polar plot and press the left mouse button.

Result

The pole or zero is placed on the pole-zero plot and the frequency response is recomputed and displayed. The mouse cursor changes back into an arrow and the x-axis label is cleared.

Notes

• Poles and zeros located off the displayed pole-zero plot can be entered using one of the coordinate entry groups described below.

"With Mouse" Property Radiobuttons

Purpose

Select whether poles and zeros placed on the pole-zero plot are complex, real or purely imaginary.

Use

1. Place the mouse cursor on the desired property and press the left mouse button.

Result

The radiobutton is selected and the chosen property is applied the next time a pole or zero is placed with the mouse.

Notes

Complex and purely imaginary poles and zeros are placed as complex-conjugate pairs. Real poles and zeros are placed as a single pole or zero on the real axis. In placing real poles and zeros, only the location of the cursor with respect to the real (horizontal) axis is important. In placing imaginary poles and zeros, only the location of the cursor with respect to the imaginary (vertical) axis is important.

Rectangular Coordinate Entry Group

Purpose

Enter a pole or zero location using rectangular (real and imaginary) coordinates.

Use

1. Enter the real and imaginary parts of one pole or zero in the complex-conjugate pair. Enter only the real or only the imaginary part of a purely real or imaginary pole or zero.
2. Select the group "X" pushbutton to place a pole (pair). Select the group "O" pushbutton to place a zero (pair).

Result

The desired pole or zero is added to the pole-zero plot and the frequency response is recomputed and displayed.

Notes

- Numeric expressions (i.e "1/2", "3*2/5", etc.,) can be used in the edit boxes.
- Blank edit boxes are assumed to be zero.

Polar Coordinate Entry Group

Purpose

Enter a pole or zero location using polar (magnitude and angle) coordinates.

Use

1. Enter the magnitude and angle from the pole-zero plot origin of one pole or zero in the complex-conjugate pair.
2. Select the group "X" pushbutton to place a pole (pair). Select the group "O" pushbutton to place a zero (pair).

Result

The desired pole or zero is added to the pole-zero plot and the frequency response is recomputed and displayed.

Notes

- Numeric expressions (i.e "pi/4", "45/2", etc.,) can be used in the edit boxes.
- Angle entry is tied to the Angle Scale checkbox. If the Angle Scale checkbox is checked, angle entry is in degrees. Otherwise, angle entry is in radians.
- Blank edit boxes are assumed to be zero.

Phase Condition Radiobuttons

Purpose

Select between a minimum, maximum or "mixed" phase transfer function.

Use

1. Place the mouse cursor on the desired condition and press the left mouse button.

Result

A new transfer function is computed and displayed according to the desired selection.

Notes

- Minimum phase systems have all their polynomial zeros located inside the unit circle. Maximum phase systems have all their polynomial zeros located outside the unit circle. "Mixed" phase systems have zeros located both inside and outside the unit circle.
- Selecting Minimum or Maximum phase does not affect the internal storage of the zero locations. Selecting "Mixed" after selecting Minimum or Maximum returns the zeros to their original location.
- The state of the Phase Condition radiobuttons affects the operation of the Print and Save pushbuttons.

Magnitude Scale Radiobuttons

Purpose

Select the scaling of the frequency response magnitude plot (logarithmic or linear).

Use

1. Place the mouse cursor on the desired scale and press the left mouse button.

Result

The frequency response if redisplayed with the desired magnitude scale.

Angle Scale Checkbox

Purpose

- Select the scaling of the frequency response phase plot (degrees or radians).
- Select the entry option for the angle edit box in the Polar Coordinate Entry group

Use

1. Place the mouse cursor on the checkbox and select or unselect as desired.

Result

When selected, the phase response is displayed in degrees and the entry option for the angle edit box is in degrees. When unselected, both of these are in radians.

Impulse Pushbutton

Purpose

Generate a plot of the first 100 points of the current filter impulse response.

Use

1. Place the mouse cursor over the pushbutton and press the left mouse button.

Result

A child graphics window is opened containing a plot of the impulse response.

Notes

- The filter's stability can be determined from the impulse response.

Print Pushbutton

Purpose

Print the filter parameters into the Matlab command prompt window.

Use

1. Place the mouse cursor over the pushbutton and press the left mouse button.

Result

Current pole and zero locations are printed in both rectangular and polar coordinates along with the transfer function coefficients in the Matlab command window.

Notes

- The state of the Phase Condition radiobuttons determines whether the parameters of a minimum, maximum or "mixed" phase system are printed.
- The filter parameters can be saved to a text file by using the diary command before printing or by cutting from the command window using the cut and paste facility, if available.
- The pole or zero locations can be saved to a variable by first saving the transfer function coefficients using the Save pushbutton and then using the Matlab roots command to compute the pole and zero locations from the transfer function polynomials.

SnapShot Pushbutton

Purpose

Recreate the current pole-zero and frequency response plots in a new graphics window as a preliminary step to obtaining hardcopy.

Use

1. Place the mouse cursor over the pushbutton and press the left mouse button.

Result

The current plots are recreated in a new graphics window. The user can then treat this window as any other Matlab graphics window. Hardcopy can be obtained using the Matlab print command.

Delete Pushbutton

Purpose

Delete a pole or zero complex-conjugate pair using the mouse.

Use

1. Place the mouse cursor over the pushbutton and press the left mouse button.
2. The mouse cursor will change into a crosshair and the x-axis label will change to "Click cursor on item to delete."
3. Move the crosshairs over one of the poles or zeros in the complex-conjugate pair and select the left mouse button.

Result

The complex-conjugate pair is deleted and the transfer function is recomputed and displayed. The mouse cursor changes back into an arrow and the x-axis label is cleared.

Notes

- Real poles and zeros are deleted one-at-a-time.

Restore Pushbutton

Purpose

Restore the original filter design.

Use

1. Place the mouse cursor over the pushbutton and press the left mouse button.

Result

If grafiltr was called with input arguments, the transfer function specified by the calling arguments is restored. All edits are lost if not previously saved using the Save pushbutton. If no input arguments were used, the Restore pushbutton has the same effect as the Clear pushbutton.

Clear Pushbutton

Purpose

Clear all poles and zeros from the filter design, effectively setting the transfer function to equal the constant 1.

Use

1. Place the mouse cursor over the pushbutton and press the left mouse button.

Result

All poles and zeros are cleared. All edits are lost if not previously saved using the Save pushbutton.

Save Pushbutton

Purpose

Saves the filter transfer function coefficients to the given variable named suffixed with an underscore and an "a" for the zero polynomial parameters and a "b" for the pole polynomial parameters.

Use

1. Place the cursor on the Save Pushbutton and click the left mouse button.
2. The Save pushbutton is replaced by an edit box. If Save has already been used, the previously entered vector name is contained in the edit box.
3. Place the cursor on the edit box and click the left mouse button. Using the keyboard, backspace over any previous entry and then type in the desired name (following standard Matlab variable name conventions).
4. Press the return key.

Result

The transfer function is saved as specified above to the Matlab workspace. The edit box is replaced by the Save pushbutton.

Notes

Once saved to the Matlab workspace, these variable can be treated like any other "a" or "b" variable returned by the Matlab Signal Processing Toolbox filter design functions.

Close Pushbutton

Purpose

Close the dodad.

Use

1. Place the mouse cursor over the pushbutton and press the left mouse button.

Result

The dodad window is closed and any global variables created by the dodad are cleared.

LD8BIT,LD16BIT,LD32BIT

Purpose

Load a file of samples stored as 8-bit signed integers.

Synopsis

```
y = ld8bit('name');
y = ld8bit('name',number)
y = ld8bit('name',number,offset)
y = ld8bit('name',0,offset);
```

Description

ld8bit('name') - Load the entire file as 8-bit signed samples.

ld8bit('name',number) - Loads the first *number* of samples starting from the beginning of the file.

ld8bit('name',number,offset) - Loads *number* of samples beginning at *offset* from the beginning of the file. If *number* = 0, loading starts from *offset* and continues to the EOF.

The LD16BIT and LD32BIT commands take the same arguments as the LD8BIT command.

Example

Datafile.dat contain four seconds of samples sampled at 8192 Hz. Load the data collected during the third second.

```
y = ld8bit('datafile.dat',2*8192+1,8192);
```

See Also

ld16bt, ld32bit

LOADSSPI,SAVESSPI

Purpose

Load data stored in the format specified by the Cyclic Spectral Analysis Software Package from Statistical Signal Processing, Inc.

Synopsis

```
y = loadsspi('name');
y = loadsspi('name','binary');
savesspi(x,'name');
```

Description

loadsspi('name') - Loads SSPI software generated data from the file *name*. Data is stored in ASCII format.

loadsspi('name','binary') - Data is stored in binary format.

savesspi(x,'name') - Saves the vector x to the file *name* in SSPI data file format. Data is written in floating point ASCII.

savesspi(x,'name','binary') - Data is stored in binary format.

ASCII SSPI format is:

first line - *type nbrsamples*
rest of lines - one or two columns of ascii numeric data

where,

type: 1 = real, 2 = complex
nbrsamples = number of samples (one sample per line).

```
2 6
3.226078e-01 2.978590e+00
5.433282e-01 8.382938e-01
3.492872e+00 9.382934e+00
3.293842e-01 2.394829e+00
4.592839e+01 2.293940e+00
6.203482e+00 2.382920e+01
```

Binary format follows the same line except data is store serially.

Reference

[1] Stephan V. Schell and Chad M. Spooner, Cyclic Spectral Analysis Software Package User's Manual, Statistical Signal Processing, Inc., 1991.

LOADVOC,SAVEVOC

Purpose

Load and save data to and from Soundblaster Voice files.

Synopsis

```
[y,fs] = loadvoc(name);
savevoc(name,x,fs);
```

Description

[y,fs] = loadvoc(name) - Loads the data samples from a Soundblaster Creative Voice File (*.voc) into *y* and stores the sampling frequency into *fs*. The function checks the file to ensure it is a version 1.10 Soundblaster file. This function only supports raw, 8-bit, unpacked voice files. The filename extension ".voc" is appended to *name* if no extension is supplied. Any mean is removed after being read from the voice file.

savevoc(name,x,fs) - Saves the data in *x* to a Soundblaster Creative Voice File (*.voc) named "name.voc". Default sampling frequency is 8192 Hz. This function saves the file as a version 1.10 file and stores the data in the raw, 8-bit unpacked format.

Example

Load a voice file, cut the sampling rate in half and store to a new file:

```
[s,fs] = loadvoc('seatsit');
y = decimate(s,2);
savevoc('slowseat',y,fs/2);
```

See Also

loadwave, savewave, readau, writeau

Reference

[1] *The Developer Kit for Sound Blaster Series*, pp 4-6 to 4-12, Creative Labs, Inc., Nov 1991.

LPERIGRM

Purpose

Display the periodogram of a signal on a logarithmic scale.

Synopsis

```
lperigrm(x)
lperigrm(x,fs)
lperigrm(x,mindb)
lperigrm(x,fs,mindb)
lperigrm(x,fs,'phase')
lperigrm(x,fs,mindb,'phase')
```

Description

`lperigrm(x)` - Computes the periodogram of *x* and displays the result on a logarithmic (decibel) scale. Only the positive frequencies are displayed. The default sampling frequency is *fs* = 8192 Hz.

`lperigrm(x,fs)` - Sets the sampling frequency to *fs*.

`lperigrm(x,mindb)` - Cuts the plot off below *mindb* for a better display of the data above. If *mindb* is positive, it is interpreted as *fs* in the above usage. Therefore, *mindb* must be negative for this usage.

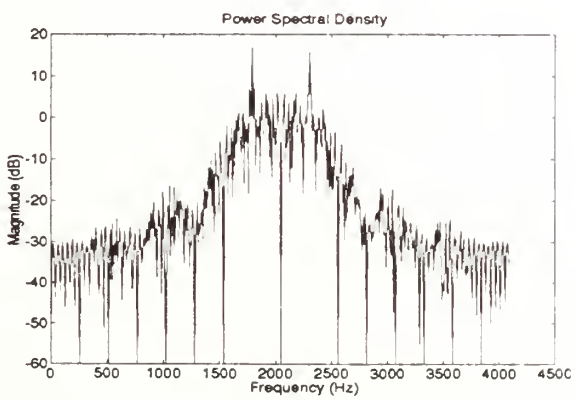
`lperigrm(x,fs,mindb)` - Used to set *fs* and *mindb* to other than their default values or to set *mindb* when *mindb* is positive.

`lperigrm(x,fs,'phase')` and `lperigrm(x,fs,mindb,'phase')` - Splits the graphics window and displays both the magnitude and phase information.

Example

Generate a frequency-shift keyed signal and display it's periodogram. Display the magnitude down to -100 dB.

```
y = bfsk(512,512,2048,0.1);
lperigrm(y,-100);
```



Algorithm

The periodogram is given by:

$$S_{x_T}(t,f) \equiv \frac{1}{T}|X_T(t,f)|^2$$

where $X_T(t,f)$ is the fourier transform of $x(t)$:

$$X_T(t,f) = \int_{t-T/2}^{t+T/2} x(u)e^{-j2\pi u}du$$

The plot magnitude is:

$$20 \log (|X_T(t,f)|)$$

Limitations

The size of the FFT used to compute the fourier transform is the closest power-of-two greater-than or equal-to the length of x. Long input vectors require longer compute times. If x is short and a finer frequency resolution is desired, the input vector x can be zero padded before calling the function.

```
lperigrm([x zeros(300,1)]);
```

See Also

wperigrm, plottime

Reference

[1] William A. Gardner, *Statistical Spectral Analysis, A Nonprobabilistic Theory*, pp. 5-7, Prentice-Hall, 1988.

LRS

Purpose

Generate a maximal-length, linear recursive sequence.

Synopsis

```
y = lrs(r,n);
y = lrs(r,n,fill,taps);
```

Description

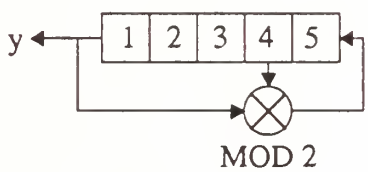
lrs(r,n) - Returns *n* bits of a binary, maximal-length, linear recursive sequence using an *r* bit shift register. Uses a randomly generated fill with randomly generated taps.

lrs(r,n,fill,taps) - Sets the initial fill and tap locations to those specified in the vectors *fill* and *taps*. These vectors may be specified by a vector containing a binary representation of the register and the taps or by a list of locations for each cell set to one and each tap location. For example, an initial fill of [1 0 1 0 1] can also be specified by [1 3 5] and taps of [1 0 0 1 0] can be specified as [1 4]. If the binary representation is specified, its length must equal *r*. If a list is specified, it largest element must be less than *r*. The output is taken from a tap off cell one. Therefore, a tap off stage one is always included whether or not it is specified.

Example

Generate 15 bits of an R5(1,4) LRS with an initial fill of all ones.

```
y = lrs(5,15,[1 1 1 1 1],[4]);
y =
        [1 0 0 1 1 0 1 0 0 1 0 0 0 0 1]
```



LS_SVD

Purpose

Compute the filter coefficients for a least-squares optimal filter using singular value decomposition.

Synopsis

```
[h,S,Px] = ar_svd(d,x,P)
```

Description

ar_svd(d,x,P) - Given the observed data sequence x, compute the filter coefficients for a least-squares optimal filter of order P, that produce the desired data sequence d from the observed data sequence. Output arguments are:

h = filter coefficients
S = sum of squared errors
Px = the projection matrix

Example

Given the desired data sequence [1 -2 3 -4 5], compute the filter coefficients of the second order least-squares filter that will produce the desired data sequence [1 -1 1 -1 1]. (Example 9.4 from [1])

```
d = [1 -1 1 -1 1];
x = [1 -2 3 -4 5];
[h,S] = ls_svd(d,x,2);
h =     1.0000
        1.0000
S =     3.5527e-015
y = filter(1,h,[1 0 0 0 0]);
y =
        1 -1 1 -1 1
```

See Also

ls_svd

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 518-528, Prentice-Hall, 1992.

LS_WHOPF

Purpose

Compute the filter coefficients for a least-squares optimal filter using the Wiener-Hopf equation.

Synopsis

```
[h,S,Px] = ar_whopf(d,x,P)
```

Description

ar_whopf(d,x,P) - Given the observed data sequence x, compute the filter coefficients for a least-squares optimal filter of order P, that will produce the desired data sequence d from the observed data sequence. Output arguments are:

h = filter coefficients
S = sum of squared errors
Px = the projection matrix

Example

Given the desired data sequence [1 -2 3 -4 5], compute the filter coefficients of the second order least-squares filter that will produce the desired data sequence [1 -1 1 -1 1]. (Example 9.3 from [1])

```
d = [1 -1 1 -1 1];
x = [1 -2 3 -4 5];
[h,S] = ls_whopf(d,x,2);
h =     1.0000
        1.0000
S =     0
y = filter(1,h,[1 0 0 0 0]);
y =
        1 -1 1 -1 1
```

Algorithm

Wiener-Hopf equation solved for h:

$$\mathbf{h} = (\mathbf{X}^{*T}\mathbf{X})^{-1}\mathbf{X}^{*T}\mathbf{d}$$

Sum of squared errors:

$$S = \mathbf{d}^{*T}\mathbf{d} - \mathbf{d}^{*T}\mathbf{X}\mathbf{h}$$

See Also

ls_svd

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 518-528, Prentice-Hall, 1992.

MINPHASE

Purpose

Return a polynomial that is in minimum-phase form.

Synopsis

```
y = minphase(P);
```

Description

minphase(P) - Returns a minimum-phase form polynomial for P if P is not minimum-phase already. If P already is minimum phase, P is simply returned. A minimum-phase polynomial is one whose roots are inside the unit circle.

Example

Return the minimum phase form of $2z^2+z+4$. The roots of this polynomial are:

$$r_{1,2} = 1.414e^{\pm j\frac{\pi}{4}}.$$

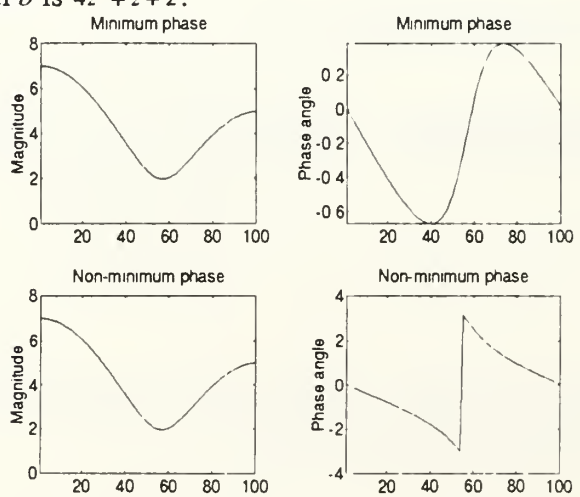
The roots of the resulting minimum phase polynomial are :

$$r_{1,2} = 0.7071e^{\pm j\frac{\pi}{4}}.$$

It can be verified using the freqz command that the magnitude of the impulse response for both these polynomials are the same and that the second polynomial has minimum phase lag.

```
b = minphase([2 1 4]);
b =     4.0000
        1.0000
        2.0000
[h,w] = freqz(b,1,50);
mag = abs(h); phase = angle(h);
semilogy(w,mag); semilogy(w,phase);
```

Polynomial b is $4z^2+z+2$.



Algorithm

The algorithm multiplies the polynomial by the following factor for each root that has a magnitude greater than 1:

$$p = p\frac{zz_0^* - 1}{z_0 - z}$$

See Also

conv, deconv, freqz, poly

References

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, pp 250-253, Prentice-Hall, 1992.

[2] Alan V. Oppenheim & Ronald W. Schafer, *Digital Signal Processing*, pp 345-353, Prentice-Hall, 1975.

MODINDEX

Purpose

Computes the modulation (deviation) index of an AM signal.

Synopsis

```
y = modindex(x);
```

Description

modindex(x) - Returns the modulation (deviation) index of *x*.

Example

Return the deviation index of a DSBLC modulated triangular wave used in the DSBLC command example.

```
x = triwave(60,30);
y = dsblc(x,0.3,1024)
m = modindex(y);
m =
        3.0000
```

Algorithm

Finds the complex envelope and from its extrema computes:

$$m = \frac{\max - \min}{\max + \min}$$

See Also

dsbsc, dsblc, envelope

NORMALEQ

Purpose

Solves a system of Normal equations used in linear predictive filtering.

Synopsis

```
[e,a] = normaleq(R)
```

Description

normaleq(R) - Returns the prediction error variance, e, and the linear predictive filter coefficients, $\begin{bmatrix} 1 & a_1 & a_2 & \dots & a_p \end{bmatrix}$.

Example

Use normaleq to solve the system of normal equations used in Example 7.1 of [1].

```
R = toeplitz([1 0.5 0.25]);
[e,a] = normaleq(R);
e =
        0.7500
a =
       -0.5000
             0
```

Algorithm

This function is a direct implementation of equation (7.23) given in [1]. Refer to [1] for a detailed discussion. In references on linear predictive filtering, these equations are often referred to as the *augmented normal equations*.

Normal equations for p = 2:

$$\begin{bmatrix} R_x[0] & R_x[1] & R_x[2] \\ R^*_x[-1] & R_x[0] & R_x[1] \\ R^*_x[-2] & R^*_x[-1] & R_x[0] \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sigma_\varepsilon^2 \\ 0 \\ 0 \end{bmatrix}$$

where σ_ε^2 is the prediction error variance and $R_x[n]$ is the auto-correlation of x at n.

Reference

[1] Charles W. Therrien, *Discrete Random Signals and Statistical Signal Processing*, eq 7.23, p 345, Prentice-Hall, 1992.

OOK,OOKMSG

Purpose

Generate an on-off keyed, bandpass signal.

Synopsis

```
y = ook(Rb,fc);
y = ook(Rb,fc,duration);
y = ook(Rb,fc,duration,fs);
y = ookmsg(Rb,fc,msg);
y = ookmsg(Rb,fc,fs,msg);
```

Description

ook(Rb,fc) - Generates one second of an on-off keyed signal with a bit rate of *Rb* bits-per-second centered at a carrier frequency, *fc*. Default sampling rate is 8192 Hz.

ook(Rb,fc,duration) - Generates *duration* seconds of the on-off keyed signal.

ook(Rb,fc,duration,fs) - Set the sampling frequency to *fs*.

ookmsg(Rb,fc,msg) or ookmsg(Rb,fc,fs,msg) - Generate a bandpass binary on-off keyed signal with a bit rate of *Rb* bits-per-second, at a carrier frequency, *fc*, sampled at sampling frequency, *fs*, and containing the message, *msg*. If *fs* is not given, *fs* defaults to 8192 Hz.

Example

Generate a 64 bit-per-second, on-off keyed signal at a carrier frequency of 2048 Hz lasting one-and-one-half seconds. Play the signal over the workstation's speaker.

```
y = ook(64,2048,1.5);
sound(y,8192);
```

Algorithm

The on-off keyed signal is generated by first creating a unipolar signal at the desired bit/sampling rates, and then applying a double-sideband, suppressed-carrier modulation to the resulting baseband signal at the desired carrier frequency.

Limitations

See limitations under unipolar.

See Also

unipolar, dsbsc, bpsk, bpskmsg, bfsk, bfskmsg

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 332-335, Macmillan, 1990.

PEAKS

Purpose

Find the peaks within a vector.

Synopsis

```
[y] = peaks(x);
[y,i] = peaks(x);
```

Description

PEAKS(x) - Returns a vector containing the magnitude of the peaks in vector *x*. A "peak" is defined as a vector member that is either greater or smaller than both of its immediate neighbors depending upon whether it is a maximum or minimum "peak". Optional output argument *i* returns a vector corresponding to the indices of the peaks returned in *y*. If the output argument *i* is not requested, output argument *y* will be the same length as x with non-peak members set to zero. If output argument *i* is requested, output argument *y* will contain only the peak values. There is a one-to-one correspondence between members of *y* and *i*.

Examples

Find the peaks of [1 5 4 2 6 7 -1 0 3].

```
x = [1 5 4 2 6 7 -1 0 3];
y = peaks(x);
y = [0 5 0 2 0 7 -1 0 3]
```

Return just the peaks and their indices.

```
[y,i] = peaks(x);
y = [5 2 7 -1 3]
i = [2 4 6 7 9]
```

Limitations

PEAKS makes extensive use of for loops and therefore will be slow when the input

PLOTTIME

Purpose

Plot the time-domain display of a signal.

Synopsis

```
plottime(x)
plottime(x,fs)
plottime(x,duration)
plottime(x,fs,duration)
plottime(x,duration,offset)
plottime(x,fs,duration,offset)
```

Description

plottime(x) - Plots the time-domain representation of *x* with a time axis scaled for a sampling frequency of 8192 Hz. Default *duration* is one second.

plottime(x,fs) - Sets the sampling frequency to *fs* Hz. The variable *fs* must be greater than or equal to 100 Hz for this usage.

plottime(x,duration) - Plot up to *duration* seconds of *x*. The *duration* must be less than 100 seconds for this usage.

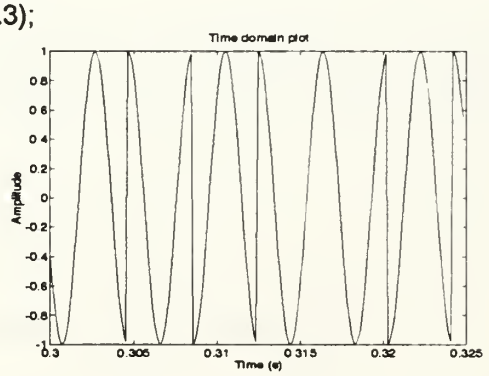
plottime(x,fs,duration) - Plots up to *duration* seconds of *x*.

plottime(x,duration,offset) and plottime(x,fs,duration,offset) - Starts the plot *offset* seconds from zero seconds.

Example

Generate one second of a binary phase-shift signal. Display 1/40 seconds of the signal beginning at 0.3 seconds.

```
y = bpsk(256,256);
plottime(y,1/40,.3);
```



See Also

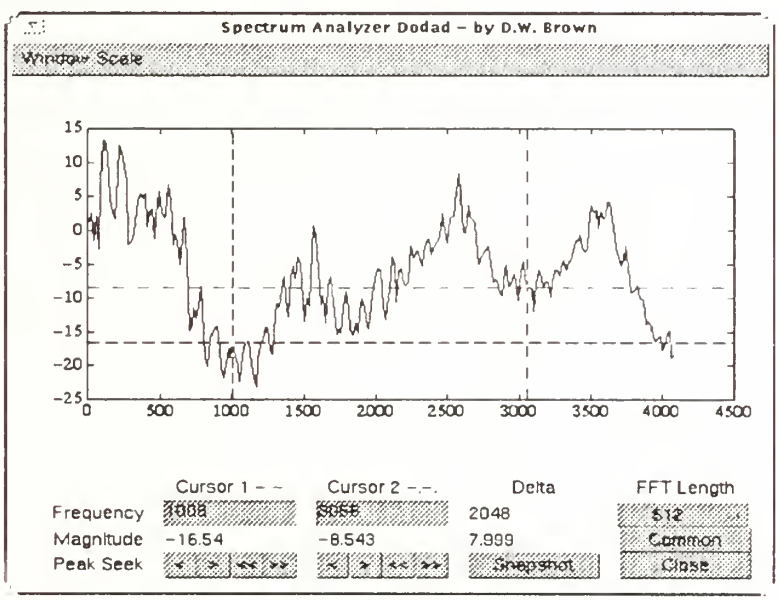
lperigrm, wperigrm

SANALYZR

The Spectrum Analyzer Dodad provides a graphical, interactive environment from which to take spectral measurements of a signal. Its operation is similar to that of the lab equipment of the same name. Two independent cursors are provided along with a digital readout of their corresponding frequencies and magnitudes and the difference in frequency and magnitude between them. The cursors may be moved from sample-to-sample using the "<" or ">" pushbuttons or from peak-to-peak (high or low) using the "<<" or ">>" pushbuttons. Single "clicking" on the spectral line with the mouse cursor moves the closest spectrum analyzer cursor to the position of the mouse cursor. A spectrum analyzer cursor can also be grabbed with the mouse and dragged to a new position. The spectrum analyzer cursors can also be moved to an specified frequency be entering the frequency in the cursor's frequency edit box and pressing return.

Starting the Spectrum Analyzer Dodad

The Spectrum Analyzer Dodad can be started in two different ways. The first is to start the dodad with a vector name as an argument. A vector is defined as a 1xN or a Nx1 Matlab variable and it must exist in the Matlab workspace before starting the Spectrum Analyzer Dodad. To analyze a vector named "myvector", the command "sanalyzr(myvector)" needs to be executed.



No output arguments are supported.

Notes

• The spectrum displayed is the output of an enhanced version of the spectrum command from the Matlab Signal Processing Toolbox named spectrm2.m which allows for user-defined windows.

• The default window is still a hanning window but the window can be changed after sanalyzr is started by selecting another window from the Window pulldown menu.
• The spectrum command uses the Welch method of spectral estimation and is used here with no window overlap. The default FFT length is 1024 points. Only the positive frequencies up to half the sampling frequency are displayed. In addition, the DC component is not displayed.
• Movement of the spectrum analyzer cursors with the mouse under the Sun operating system requires holding the mouse button down until the cursor movement is complete. Also, after dragging a cursor with the mouse, the cursor must be held down until the cursor position stabilizes. This applies to both versions 4.0a and 4.1 of Matlab running under Sun Open Windows. Under Microsoft Windows, a quick click is all that is required to move the closest spectrum analyzer cursor to the mouse cursor location and no pause is required after dragging a spectrum analyzer cursor to a new position.

SAWWAVE

Purpose

Generate a baseband, sawtooth wave. This function can also be used to generate a ramp.

Synopsis

```
y = sawwave(fb);
y = sawwave(fb,'antipodal');
y = sawwave(fb,duration);
y = sawwave(fb,duration,'antipodal');
y = sawwave(fb,duration,fs);
y = sawwave(fb,duration,fs,,'antipodal');
```

Description

sawwave(fb) - Generates one second of a baseband, sawtooth wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

sawwave(fb,duration) - Generates *duration* seconds of a baseband, sawtooth wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

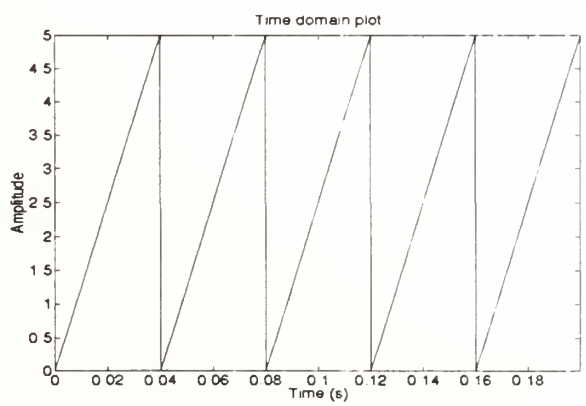
sawwave(fb,duration,fs) - Generates *nbrcycles* of a baseband, sawtooth wave at cycle frequency, *fb*, and sampling frequency, *fs*.

The argument 'antipodal' changes the amplitude range of the output waveform from [0,1] to [-1,1].

Examples

Generate 30 cycles of a sawtooth wave with a cycle frequency of 25 Hz sampled at 8000 Hz, varying between 0 and 5 volts.

```
x = 5 * sawwave(25,30/25,8000);
plottime(x,8000,0.2);
```



Generate a ramp from -1 to 3 with 100 samples:

```
y = 4 * sawwave(1,1,100) - 1;
```

Limitations

The waveform is created by first creating a single cycle of the waveform and then repeating this cycle the required number of times. Hence, when the cycle rate is not evenly divisible into the sampling frequency, truncation errors occur. The length of the output vector, *y*, can be computed by the formula:

$$length = floor(duration \cdot f_b) \cdot floor\left(\frac{f_s}{f_b}\right)$$

The exact cycle frequency can be computed from:

$$f_c = \frac{1}{floor\left(\frac{f_s}{f_b}\right)}$$

See Also

sqwave, triwave, coswave, sinwave

SETSNR

Purpose

Mix two vectors (one representing a signal, one representing noise), such that the output signal's total bandwidth signal-to-noise ratio is set to the specified value.

Synopsis

```
[y,sigma] = setsnr(signal,SNR);
[y,sigma] = setsnr(signal,noise,SNR);
```

Description

SETSNR(signal,noise,SNR) - Returns the additive signal such that the total bandwidth signal-to-noise ratio is equal to *SNR* decibels. The term "total bandwidth" implies the noise power in all frequencies from 0 to *fs/2* is used in the computation of the SNR. Sigma is the parameter σ described below.

SETSNR(signal,SNR) - Gaussian white noise is added by the function.

Example

Create a frequency-shift keyed signal and then add Gaussian white noise such that the total bandwidth SNR is 15 decibels.

```
x = bfsk(64,2048);
y = setsnr(x,randn(length(x),1),15);
```

Algorithm

The output signal is defined by

$$y[n] = s[n] + \sigma \cdot \omega[n]$$

for which σ is computed such that

$$SNR = 10 \log \left[\frac{Var(s[n])}{\sigma^2 Var(\omega[n])} \right]$$

where *SNR* is the desired signal-to-noise ratio. The amplitude of the noise is adjusted before the noise is added to produce the desired *SNR*. In the case of Gaussian white noise, relationship between σ and the noise power spectrum:

$$\sigma^2 = \frac{N_0}{2}$$

See Also

setsnrbw

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 103-105, Macmillan, 1990.

SETSNRBW

Purpose

Mix two vectors (one representing a signal, one representing noise), such that the output signal's noise-equivalent (in-band) signal-to-noise ratio is set to the specified value.

Synopsis

```
[y,sigma] = setsnrbw(signal,SNR,fc,bw);
[y,sigma] = setsnrbw(signal,noise,SNR,fc,bw,fs);
```

Description

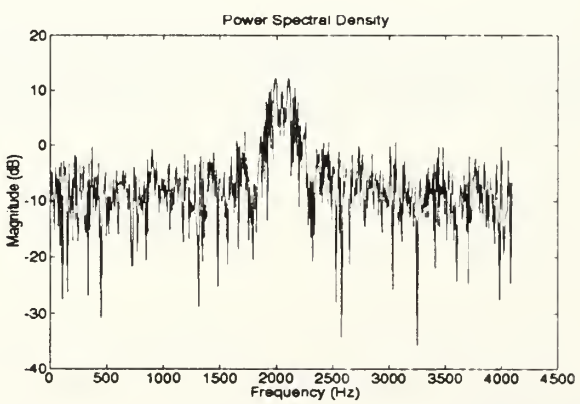
SETSNRBW(signal,noise,SNR,fc,bw,fs) - Returns the additive signal such that the noise-equivalent bandwidth (in-band) signal-to-noise ratio is equal to *SNR* decibels. The term "noise-equivalent bandwidth" implies the noise power only in frequencies within the signal bandwidth is used in computing the SNR. The signal bandwidth is centered at f_c and is equal to 2**bw*. This definition requires the value actually supplied for *bw* be one-half the desired bandwidth specification (see example). The sampling frequency, *fs*, is required to find the power spectral density of the noise. Sigma is the parameter σ described below.

SETSNRBW(signal,SNR,fc,bw) - Gaussian white noise is added by the function and the sampling frequency defaults to 8192 Hz.

Example

Create a coherent, frequency-shift keyed signal and then add Gaussian white noise such that the noise-equivalent bandwidth SNR is 15 decibels. Use the distance from the center frequency to the first null as one-half the bandwidth.

```
x = bpsk(256,2048);
y = setsnrbw(x,15,2048,256,8192);
lpenigrm(y);
```



Algorithm

The output signal is defined by

$$y[n] = s[n] + \sigma \cdot \omega[n]$$

for which σ is computed such that

$$SNR = 10 \log \left[\frac{Var(s[n])}{\sigma^2 S_{BW}(\omega[n])} \right]$$

where *SNR* is the desired signal-to-noise ratio. The amplitude of the noise is adjusted before the noise is added to produce the desired *SNR*. In the case of Gaussian white noise, the relationship between σ and the power spectral density of the noise is:

$$\sigma^2 = \frac{N_0}{2}$$

See Also

setsnr

Reference

[1] Leon W. Couch, *Digital and Analog Communication Systems*, pp. 103-105, Macmillan, 1990.

SP_STENG,SP_STMAG,SP_STZCR

Purpose

Compute the short-time energy, magnitude and zero-crossing curves of speech signals.

Synopsis

```
[y,t] = sp_steng(x,frame,overlap,fs,'window');
[y,t] = sp_stmag(x,frame,overlap,fs,'window');
[y,t] = sp_stzcr(x,frame,overlap,fs,'window');
```

Description

SP_STENG(x,frame,overlap,fs,'window') - Compute the short-time energy of *x* using a window size of *frame* length and a percentage *overlap* between successive windows using a "window" prefiltering data window. Available windows are 'rectangular', 'hamming', 'hanning', 'blackman' and 'bartlett'. A rectangular window is the default if the "window" argument is not given. The output arguments are:

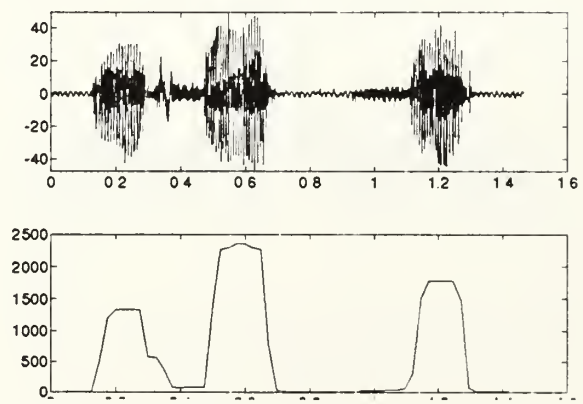
y - short-time function curve.
t - time indices for each corresponding sample in y.

SP_STMAG(x,frame,overlap,fs,'window') - Compute the short-time magnitude.
SP_STMAG(x,frame,overlap,fs,'window') - Compute the short-time zero-crossing rate.

Examples

Compute and display the short-time energy using a 30 millisecond frame, a 20% frame overlap, a Hamming window and a median smoothing filter. Note the steps necessary to plot the graphs on the same scale and aligned in time.

```
[y,t] = sp_steng(x,0.03,20,fs,'hamming');
yy = medsmooth(y,7);
subplot(2,1,1);plot((0:length(x)-1)/fs,x);
subplot(2,1,1);plot(t,yy)
```



Algorithm

Short-time energy:

$$E_n = \sum_{m=-\infty}^{\infty} |[x(m)\, w(n-m)]|^2$$

Short-time magnitude:

$$M_n = \sum_{m=-\infty}^{\infty} |x(m)|\, w(n-m)$$

Short-time zero crossings:

$$Z_n = \sum_{m=-\infty}^{\infty} |\mathrm{sgn}[x(m))] - \mathrm{sgn}[x(m-1)]|\, w(n-m)$$

See Also

avsmooth, mdsmooth

Reference

L. R. Rabiner & R. W. Schafer, *Digital Processing of Speech Signals*, pp 120-130, Prentice Hall, 1978.

SQWAVE

Purpose

Generate a baseband, unipolar [0,1] or antipodal [-1,1] square wave.

Synopsis

```
y = sqwave(fb);
y = sqwave(fb,'antipodal');
y = sqwave(fb,duration);
y = sqwave(fb,duration,'antipodal');
y = sqwave(fb,fs,nbrcycles);
y = sqwave(fb,fs,nbrcycles,'antipodal');
```

Description

sawwave(fb) - Generates one second of a baseband, sawtooth wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

sawwave(fb,duration) - Generates *duration* seconds of a baseband, sawtooth wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

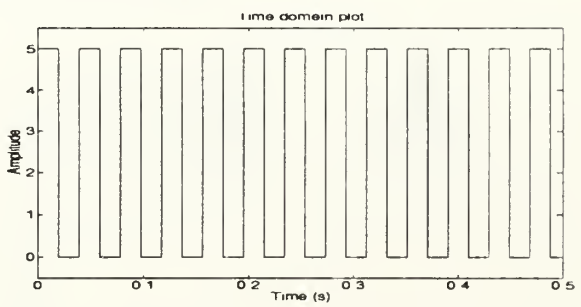
sqwave(fb,duration,fs) - Generates *duration* of a baseband unipolar [0,1] square wave at cycle frequency, *fb*, and sampling frequency, *fs*.

The argument 'antipodal' changes the amplitude range of the output waveform from [0,1] to [-1,1]. Note: If the cycle frequency, *fb*, and the sampling frequency, *fs*, are both equal to one, the resulting vector will contain alternating values of [0,1] or [-1,1].

Examples

Generate 30 cycles of a square wave with a cycle frequency of 25 Hz sampled at 8000 Hz, varying between 0 and 5 volts.

```
x = 5 * sqwave(25,30/25,8000);
```



Generate 30 bits of a 50 bit-per-second signal sampled at 8000 Hz. Use this baseband signal as the modulating signal input to dsbsc to generate a phase-shift keyed signal.

```
x = sqwave(50/2,2*30/50,8000,'antipodal');
y = dsbsc(x,2000,8000);
```

Generate a vector with 100 elements alternating between 1 and 0.

```
y = sqwave(1,1,100)
y =
        1
        0
        1
        0
        1
        0
        1
  (and so forth)
```

Limitations

The waveform is created by first creating a single cycle of the waveform and then repeating this cycle the required number of times. Hence, when the cycle rate is not evenly divisible into the sampling frequency, truncation errors occur. The length of the output vector, *y*, can be computed by the formula:

$$length = floor(duration \cdot f_b) \cdot floor\left(\frac{f_s}{f_b}\right)$$

The exact cycle frequency can be computed from:

$$f_c = \frac{1}{floor\left(\frac{f_s}{f_b}\right)}$$

See Also

antpodal, unipolar, triwave, sawwave

STR2MASC,MASC2STR

Purpose

Convert a string into its binary ASCII representation as generated by a modem.

Synopsis

```
y = str2masc('string')
y = str2masc('string',databits,'parity',stopbits)
y = masc2str(x)
y = masc2str(x,databits,'parity',stopbits);
```

Description

str2masc('string') - Converts *string* to a vector of 1's and 0's representing raw, eight-bit ASCII coding.

str2masc('string',databits,'parity',stopbits) - Converts *string* to a vector of 1's and 0's representing modem encoded ASCII. The number of *databits* can be 7 or 8. *Parity* can be 'n' for none, 'o' for odd, or 'e' for even. The number of *stopbits* can be 1 or 2. One start bit is always used. Valid combinations are:

7n1, 7e1, 7o1, 7n2, 7e2, 7o2, 8n1, 8n2

The output argument, *y*, is a *N by 1* vector where *N* is based upon the length of *string* and the combination of the parameters *bits*, *parity*, and *stopbits*.

masc2str(x) - Converts a vector of 1's and 0's representing raw eight-bit ASCII coding into a string.

masc2str(x,databits,'parity',stopbits) - Converts a vector of 1's and 0's representing the binary modem ASCII into a string. Valid argument combinations are the same as those listed above for str2masc. Parity and start/stop bit errors are detected and shown in the output string by the following special characters:

*, startbit error, ~, first stopbit error
', second stopbit error, +, parity error

Parity errors take precedence over start/stop bit errors in the output string.

Examples

Generate the binary representation of "BZ" sent at 7 bits-per-character with even parity and 2 stop bits. Convert back to a string.

```
y = str2masc('BZ',7,'e',2);
y =
        [1 1 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1]
x = masc2str(y,7,'e',2);
x =
        BZ
```

TRIWAVE

Purpose

Generate a baseband, triangular wave.

Synopsis

```
y = triwave(fb);
y = triwave(fb,'antipodal');
y = triwave(fb,duration);
y = triwave(fb,duration,'antipodal');
y = triwave(fb,duration,fs);
y = triwave(fb,duration,fs,'antipodal');
```

Description

triwave(fb) - Generates one second of a baseband, triangular wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

triwave(fb,duration) - Generates *duration* seconds of a baseband, triangular wave at cycle frequency, *fb*, and sampling frequency of 8192 Hz.

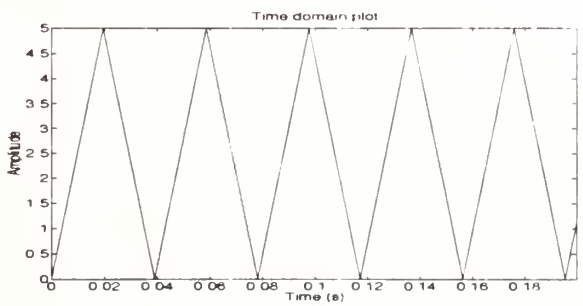
triwave(fb,duration,fs) - Generates *nbrcycles* of a baseband, triangular wave at cycle frequency, *fb*, and sampling frequency, *fs*.

The argument 'antipodal' changes the amplitude range of the output waveform from [0,1] to [-1,1].

Example

Generate 30 cycles of a triangular wave with a cycle frequency of 25 Hz sampled at 8000 Hz, varying between 0 and 5 volts.

```
x = 5 * triwave(25,30/25,8000)
```



Limitations

The waveform is created by first creating a single cycle of the waveform and then repeating this cycle the required number of times. Hence, when the cycle rate is not evenly divisible into the sampling frequency, truncation errors occur. The length of the output vector, *y*, can be computed by the formula:

$$length = floor(duration \cdot f_b) \cdot floor\left(\frac{f_s}{f_b}\right)$$

The exact cycle frequency can be computed from:

$$f_c = \frac{1}{floor\left(\frac{f_s}{f_b}\right)}$$

A "mathematically perfect" triangular wave is generated such that successive cycles fit perfectly. Without getting into a long discussion, the bottom line is that when the number of samples-per-cycle is odd, the maximum value of *y* will not be *exactly* 1, as occurs when the number of samples-per-cycle is even. In addition, the last cycle does not return *exactly* to zero in either case (the beginning of the *next* cycle would have started at zero). The "floor" term in the above equation is the number of samples-per-cycle.

See Also

sqwave, sawwave

UNIPOLAR

Purpose

Generate a baseband, unipolar [0,1] signal.

Synopsis

```
y = unipolar(fb,msg
y = unipolar(fb,msg,fs)
```

Description

unipolar(fb,msg) - Generates a baseband unipolar signal at bit rate, *fb*. Sampling frequency defaults to 8192 Hz. If *msg* is a scalar, a *message* will be generated as a random binary sequence of length *msg* with Pr(0)=Pr(1)=0.5. If *msg* is a vector, it must be a vector of 0's and 1's. The length of the output vector, *y*, can be computed by the formula:

$$length = nbrbaud \cdot floor\left(\frac{f_s}{f_b}\right)$$

unipolar(fb,msg,fs) - Sets the sampling frequency to *fs*.

Note: If the bit rate, *fb*, and the sampling frequency, *fs*, are both equal to one, the resulting vector will contain alternating values of [0,1].

Example

Generate 30 bits of a 50 bit-per-second signal sampled at 8000 Hz. Use this baseband signal as the modulating signal input to dsbsc to generate an on-off keyed signal.

```
x = unipolar(50,30,8000);
y = dsbsc(x,2000,8000);
```

Limitations

- The probability 1 occurs is not exact for a small number of bits.
- If the sampling frequency is not an exact multiple of the bit rate, unexpected results may occur. For example: unipolar(9,18,20) produces an output of 36 samples vice the expected 40 samples (i.e. 18 / 9 = 2 seconds at 20 samples-per-second). This occurs since *floor(fs/fb)* = 2 samples-per-bit.

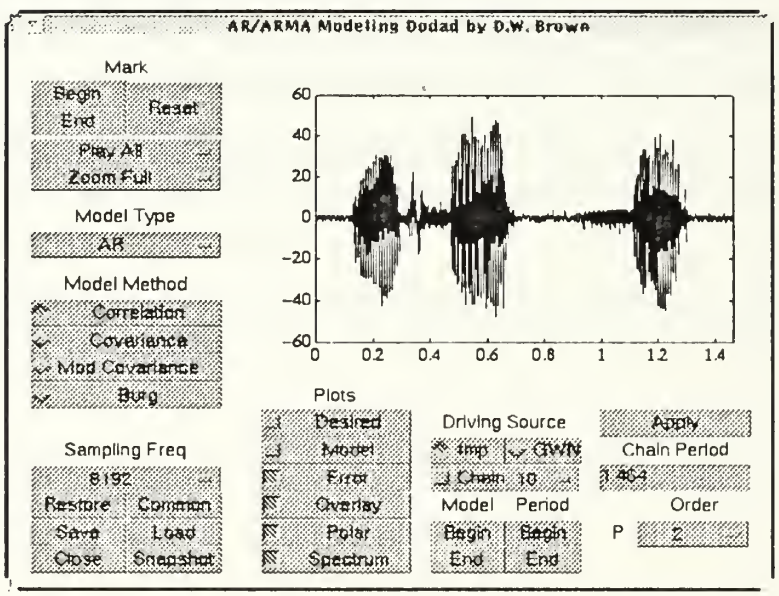
See Also

antpodal, sqwave

VECTARMA

The Vector ARMA Dodad provides an interactive, graphical environment within Matlab to model Auto-Regressive (AR) and Auto-Regressive/Moving Average (ARMA) processes. With the Vector ARMA Dodad, you can:

- Model AR processes using the autocorrelation, covariance, modified covariance or Burg methods.
- Model ARMA process using the Prony, Durbin or Shank methods.
- Generate minimum-phase ARMA models.
- Drive the model with an impulse function or Gaussian white noise.
- Plot the impulse response.
- Plot the error signal.
- Generate a chain of models. This feature is useful in modeling parts of speech.
- Play the model and error signal on the workstation' speaker.



Starting the Vector Filter Dodad

The Vector ARMA Dodad can be started in one of two different ways. The first one is to start the dodad with a vector name as an argument. A vector is defined as a 1xN or a Nx1 Matlab variable and it must exist in the Matlab workspace before starting the Vector ARMA Dodad. To model a vector named "myvector", the command "vectarma(myvector)" needs to be executed. Supplying the command with a second argument of "1" starts the dodad with a sampling frequency of one.

No output arguments are supported. The model must be saved using the save push-button as described under common controls.

The second way to start the dodad is to execute the "vectarma" command without

any arguments. This option can be chosen only after the vectedit or vectfilt dodad has been used to modify a vector. Every time a vector is modified using these dodads, the modified vector is saved to a common vector in the global area of the Matlab workspace. When no vector argument is supplied, this common vector is loaded into the vectarma dodad.

Vector ARMA Dodad Specific Controls

In addition to controls discussed under common controls, the following controls are specific to the Vector ARMA Dodad (vectarma). For the most part, these controls simply set the filter parameters. No filtering actually takes place until the Apply pushbutton is selected. Additionally, not all of the following controls are available at any one time. Only controls pertinent to the currently selected model type are displayed.

Model Type Popupmenu

Purpose

Selects between AR and ARMA modeling.

Use

1. Place the cursor on the Model Type popupmenu and click the left mouse button once to open.
2. Place the cursor on the desired model type (AR or ARMA) and click the left mouse button.

Result

The desired model type is selected and displayed on the popupmenu. Additional controls are added or removed dependent upon the model type chosen.

Model Method Radiobuttons

Purpose

Selects the modeling method to be implemented.

Use

1. Place the cursor on the radiobutton of the desired modeling method and click the left mouse button once to select.

Result

The radiobutton of the desired modeling method is selected and all others are turned off.

Min Phase Checkbox

Purpose

Turns minimum phase ARMA modeling on or off.

Use

1. Place the cursor on the checkbox and click the left mouse button until the checkbox is turned on or off as desired.

Result

The ARMA model transfer function polynomials are forced to a minimum phase condition when generated.

Notes

- This control is only available when the current modeling method is set to ARMA.
- A minimum-phase model may or may not best fit the modeled data.

P Order Popupmenu

Purpose

Sets the order of the AR model (poles).

Use

1. Place the cursor on the order popupmenu and click the left mouse button once to open the menu.
2. Place the cursor on the desired order (1,2,3,4,6,8,10,12,14) and click the left mouse button. If the desired order is not one of the default options, it can be entered manually by selecting the "User" option. In this case, an edit box appears in place of the popupmenu. Enter the desired order into the edit box and press return.

Result

The AR order is set.

Q Order Popupmenu

Purpose

Sets the order of the MA model (zeros).

Use

1. Place the cursor on the order popupmenu and click the left mouse button once to open the menu.
2. Place the cursor on the desired order (1,2,3,4,6,8,10,12,14) and click the left mouse button. If the desired order is not one of the default options, it can be entered manually by selecting the "User" option. In this case, an edit box appears in place of the popupmenu. Enter the desired order into the edit box and press return.

Result

The MA order is set.

Notes

This control is only available when the current modeling method is set to ARMA.

Plots Checkboxes

Purpose

Selects the plots to be displayed.

Use

1. Place the cursor over the desired plot and click the left mouse button to turn the plot on or off.

Result

The desired plots are displayed when the model is generated.

Driving Source Radiobuttons

Purpose

Chooses between an impulse and Gaussian white noise to drive the model.

Use

1. Place the cursor over the desired driving source and click the left mouse button to select.

Result

The desired source is used when the model is applied.

Model Mark Start/Mark End Pushbuttons

Purpose

Marks the beginning and end of a portion of the vector to be used in generating the model.

Use

1. Select desired mark pushbutton (Begin or End).
2. After selection, the cursor changes to a crosshair and the plot title changes to either "Mark start of model with cursor" or "Mark end of model with cursor" as appropriate.
3. Move the cursor to the desired mark location and click the left mouse button. When placing the mark, only the location on the time axis is of importance. Placing a mark past either end of the plot sets the mark to the first or last value displayed on the horizontal axis.

Result

After placing a mark, the cursor changes back into an arrow and the plot title is cleared. The model-begin mark is displayed as a vertical dashed line and the model-end mark is displayed as a vertical dotted line.

Notes

The default option uses the entire vector to generate the model.

Chain Period Mark Start/Mark End Pushbuttons

Purpose

Marks the beginning and end of the Chain period.

Use

1. Select desired mark pushbutton (Start or End).
2. After selection, the cursor changes to a crosshair and the plot title is changed to either "Mark start of period with cursor" or "Mark end of period with cursor" as appropriate.
3. Move the cursor to the desired mark location and click the left mouse button. When placing the mark, only the location on the time axis is of importance. Placing a mark past either end of the plot sets the mark to the first or last value displayed on the horizontal axis.

Result

After placing a mark, the cursor changes back into an arrow and the plot title is cleared. The period-begin mark is displayed as a vertical dashed line and the period-end mark is displayed as a vertical dotted line.

Notes

- The default option uses the entire vector length as the period.
- The period-end cannot extend past the length of the vector from the period begin mark.

• The period begin mark is set coincidental with the model begin mark by the Model Begin pushbutton.

Chain Checkbox

Purpose

Turns the generation of a chain of models on or off.

Use

1. Place the cursor on the Chain checkbox and click the left mouse button to turn the chain on or off.

Result

The model is chained the number of times specified by the Chain Length popupmenu when the Chain checkbox is checked.

Chain Length Popupmenu

Purpose

Sets the number of times the model is repeated in the chain.

Use

1. Place the cursor on the Chain Length popupmenu and click the left mouse button once to open the menu.
2. Place the cursor on the desired chain length (10,20,30,40,50 or 60) and click the left mouse button. If the desired chain length is not one of the default options, the chain length can be entered manually by selecting the "User" option. In this case, an edit box appears in place of the popupmenu. Enter the desired chain length into the edit box and press return.

Result

The desired chain length is set.

Notes

This feature is useful for modeling parts of speech.

Chain Period Edit Box

Purpose

• Displays the time between the Period Markers.
• Allows the operator to manually enter the chain period.

Use

1. Place the cursor over the edit box and click the left mouse button.
2. Enter the desired chain period length and press return.

Result

The chain period is displayed in the edit box.

Notes

The period end cannot extend past the length of the vector from the period-begin mark. When setting the period past the length of the vector, the period is adjusted so that the period-end mark is set to the end of the vector.

Play Desired, Play Model, Play Error Popupmenu Menu Items

Purpose

Sends the desired data vector, the model vector or the error vector to the worksta-

tion's audio output.

Use

1. Place the cursor on the Play popupmenu and click the left mouse button to open the menu.
2. Select the Play All menu item.

Result

The data is sent to the audio output.

Note

This option is ideal for testing speech models.

Save Pushbutton

Purpose

Saves the modeled data, model and error signals to the Matlab workspace under a new variable name, if a model was generated. In addition, the model parameters are saved to the given variable named suffixed with an underscore and an "a" for the AR parameters and a "b" for the MA parameters.

Use

1. Place the cursor on the Save Pushbutton and click the left mouse button.
2. The Save and Load pushbuttons are replaced by an edit box. If save has already been used, the previously entered vector name is contained in the edit box.
3. Place the cursor on the edit box and click the left mouse button. Using the keyboard, backspace over any previous entry and then type in the desired vector name (following standard Matlab variable name conventions).
4. Press the return key.

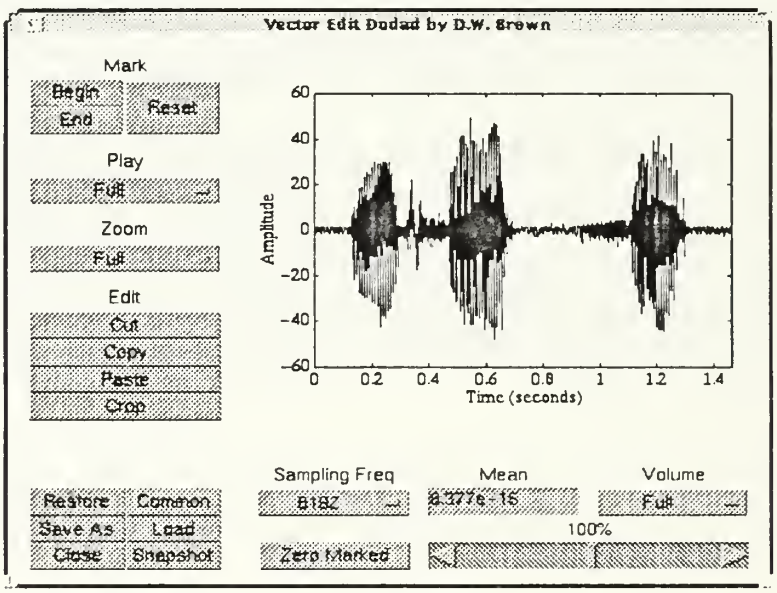
Result

The data is saved into an Nx4 matrix where the first column is the time indices, the second column is the modelled data, the third column is the model and the fourth column is the error signal.

VECTEDIT

The Vector Edit Dodad provides an interactive, graphical environment within Matlab to edit vectors. This is ideally suited for editing vectors that represent digitized signals such as speech. With the Vector Edit Dodad, you can:

- Cut unwanted sections of a signal.
- Crop unwanted sections of a signal.
- Rearrange sections of a signal by cut and paste.
- Adjust the amplitude (volume) of all or a portion of a signal.
- Introduce periods of silence into a signal.
- Simply zoom in on portions of a signal for a better look.



Starting the Vector Edit Dodad

The Vector Edit Dodad can be started in one of two ways. The first one is to start the dodad with a vector name as an argument. A vector is defined as a 1xN or a Nx1 Matlab variable and it must exist in the Matlab workspace before starting the vector edit dodad. To edit a vector named "myvector", the command "vectedit(myvector)" needs to be executed.

No output arguments are supported. Edited vectors must be saved using the save pushbutton as described under common controls.

The second way to start the Vector Edit Dodad is to execute the "vectedit" command without any arguments. This can be done only after the vectedit or vectfilt dodad has been used to modify a vector. Everytime a vector is modified using these dodads, the modified vector is saved to a common vector in the global area of the Matlab workspace. When no vector argument is supplied, this common vector is loaded into the Vector Edit Dodad.

Vector Edit Dodad Specific Controls

In addition to controls discussed in the Common Controls chapter, the following controls are specific to the vector edit dodad (vectedit).

Cut Pushbutton

Purpose

Removes a marked section of the vector placing the cut section into the cut and paste buffer.

Use

1. Mark the section to be cut using the Mark Start and Mark End pushbuttons.
2. Place the cursor on the Cut pushbutton and click the left mouse button.

Result

The marked section is cut and the display is redrawn.

Notes

• The removed portion can be restored by immediately selecting the Cut pushbutton a second time. This option is void if either of the marks are moved before selecting the Cut pushbutton for the second time or any other editing options is performed.

• If the beginning and end marks are located at the beginning and end of the vector (i.e. the entire vector is marked), selecting the Cut pushbutton has no effect.

• The common vector is modified.

Copy Pushbutton

Purpose

Copies a marked section of the vector, placing the copied portion into the cut and paste buffer.

Use

1. Mark the section to be copied using the Mark Start and Mark End pushbuttons.
2. Place the cursor on the Copy pushbutton and click the left mouse button.

Result

The copied section is placed into the cut and paste buffer.

Paste Pushbutton

Purpose

Inserts the contents of the cut and paste buffer into the vector at a point selected with the mouse.

Use

1. Place the cursor on the Paste pushbutton and click the left mouse button.
2. The cursor changes into a crosshair and the display title changes to "Mark paste insertion point with cursor."
3. Move the cursor to the desired insertion point and click the left mouse button.

Result

The contents of the cut and paste buffer are inserted into the vector and the vector display is redrawn.

Notes

• If the cut and paste buffer is empty, no action is performed (a cut or copy has to

be performed first to fill the cut and paste buffer).

• The contents of the cut and paste buffer are not cleared. Multiple copies of the cut and paste buffer can be inserted by reselecting the Paste Pushbutton.

• The common vector is modified.

Crop Pushbutton

Purpose

Deletes sections of the vector outside the currently mark region.

Use

1. Mark the region to be retained using the Mark Start and Mark End pushbuttons.
2. Place the cursor on the Crop pushbutton and click the left mouse button.

Result

The sections of the vector outside the marked region are deleted.

Notes

• The deleted sections are not saved.

• The cut and paste buffer is not modified.

• The common vector is modified.

Mean User Edit Box

Purpose

• Display the current vector's mean value.

• Remove the mean.

• Change the mean (if the vector represents a voltage, this is equivalent to changing the DC offset).

Use

Remove the mean:

1. Select the edit box by clicking the left mouse button while the cursor is located over the edit box.
2. Enter zero into the edit box and press return.

Change the mean:

1. Select the edit box by clicking the left mouse button while the cursor is located over the edit box.
2. Enter the desired offset value into the edit box and press return.

Result

The mean is adjusted and the display is redrawn.

Notes

• After the mean is changed, the mean is recomputed and redisplayed in the edit box. Note that after removing the signal mean, the resulting value may not be exactly zero due to round-off errors.

• The common vector is modified.

Zero Marked Pushbutton

Purpose

Zeroes out the vector in the marked section. This option is useful for creating absolute periods of silence.

Use

1. Mark the section to be zeroed out using the Mark Start and Mark End pushbut-

tons.

2. Place the cursor on the Zero Marked pushbutton and click the left mouse button.

Result

The marked vector section is set to zero and the display is redrawn.

Notes

The common vector is modified.

Volume Full/Marked Popupmenu and Volume Slider

Purpose

Adjust the amplitude (volume) of the vector or a marked section 0% to 200% of it's current amplitude.

Use

1. If desired, mark a section to be adjusted using the Mark Start and Mark End pushbuttons.
2. Adjust the volume slider to the desired amplitude change percent.
3. Place the cursor on the Volume popupmenu and click the left mouse button once to open the popupmenu.
4. Place the cursor on the desired function (Volume Full or Volume Marked) and click the left mouse button.

Result

The amplitude is adjusted as requested and the display is redrawn.

Notes

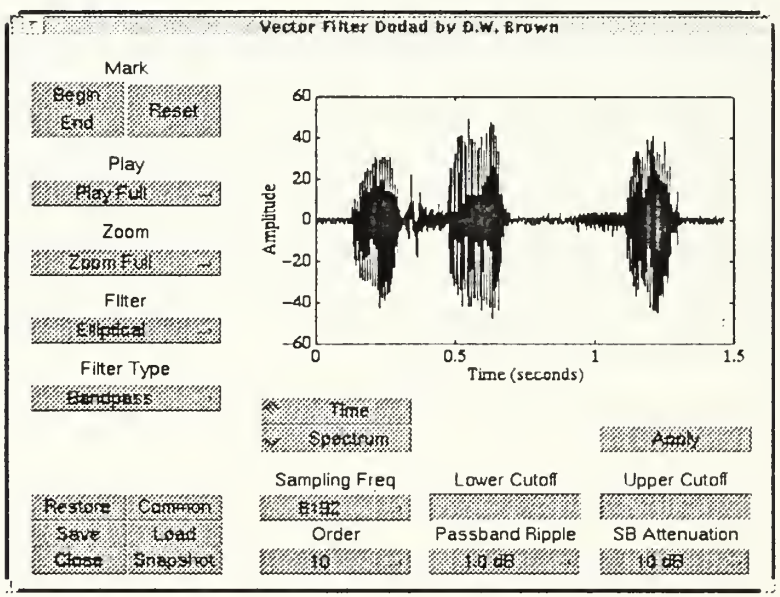
• If the vector is to be saved in an audio file format, care must be taken to ensure the new range of values do not exceed the limits of the target format. For example, Microsoft Window "wave" files and Soundblaster "voice" files usually contain only 8-bit values (-128 to +127). Values outside this range are clipped when saved to the audio file. The amplitude can be properly adjusted to range between +127 and -127 by the command ">> myvector = 127 * myvector / max(myvector);".

• The common vector is modified.

VECTFILT

The Vector Filter Dodad provides an interactive, graphical environment within Matlab to filter vectors. This is ideally suited for filtering vectors that represent digitized signals such as speech or digital bandpass signals. With the Vector Filter Dodad, you can:

- Perform lowpass, highpass, bandpass and stopband filtering.
- Filter using a Finite Impulse Response filter.
- Filter using Butterworth, Chebychev or Elliptical Infinite Impulse Response (IIR) filters.
- Vary filter parameters such as FIR window, filter order and stopband attenuation.
- View the filter transfer function before applying.
- Inspect the signal by zooming in on the time domain display.
- Inspect the signal by zooming in on the power spectral density display.
- Determine the audible effects of filtering by playing filtered signals on the speaker.



Starting the Vector Filter Dodad

The Vector Filter Dodad can be started in one of two ways. The first one is to start the dodad with a vector name as an argument. A vector is defined as a 1xN or a Nx1 Matlab variable and it must exist in the Matlab workspace before starting the Vector Filter Dodad. To filter a vector named "myvector", the command "vectfilt(myvector)" needs to be executed.

No output arguments are supported. Filtered vectors must be saved using the Save pushbutton as described under common controls.

The second way to start the filter dodad is to execute the "vectfilt" command without any arguments. This can be done only after the vectedit or vecttime dodad has been used to modify a vector. Every time a vector is modified using these dodads, the modified vector is saved to a common vector in the global area of the Matlab workspace. When no vector argument is supplied, this common vector is loaded into the Vector Filter Dodad.

Vector Filter Dodad Specific Controls

In addition to controls discussed under *common controls*, the following controls are specific to the Vector Filter Dodad (vectfilt). For the most part, these controls simply set the filter parameters. No filtering actually takes place until the apply pushbutton is selected. Additionally, not all of the following controls are available at any one time. Only controls pertinent to the currently selected filter and filter type are displayed. For example, there is no need to set an upper cutoff frequency when a low or highpass filter type is selected.

Filter Popupmenu

Purpose

Selects the filtering method to be used.

Use

1. Place the cursor on the Filter popupmenu and click the left mouse button once to open.
2. Place the cursor on the desired filter (FIR, Butterworth, Chebychev Type 1, Chebychev Type 2, or Elliptical) and click the left mouse button.

Result

The desired filtering method is selected and displayed on the popupmenu. Additional controls are added or removed dependent upon the filtering method chosen.

Notes

The filters are implemented using functions from the Matlab Signal Processing Toolkit which is required to use this dodad.

Filter Type Popupmenu

Purpose

Selects the filter type to be implemented.

Use

1. Place the cursor on the Filter Type popupmenu and click the left mouse button once to open.
2. Place the cursor on the desired filter type (Lowpass, Highpass, Bandpass, or Stopband) and click the left mouse button.

Result

The desired filtering method is selected and displayed on the popupmenu. Additional controls are added or removed dependent upon the filter type chosen.

Window Popupmenu

Purpose

Selects the type of data prefiltering window to be applied when applying a FIR fil-

tering.

Use

1. Place the cursor on the window popupmenu and click the left mouse button once to open.
2. Place the cursor on the desired window (Rectangular, Triangular, Hamming, Hanning, Blackman, or Bartlett) and click the left mouse button.

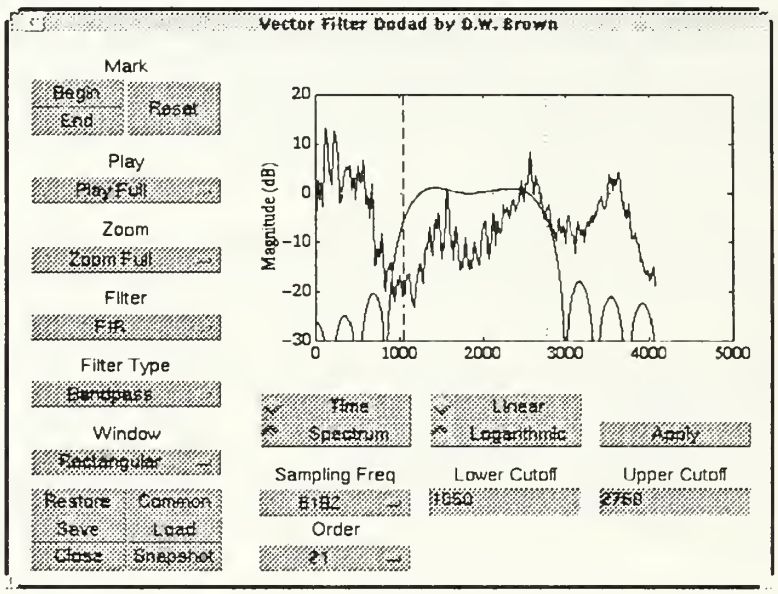
Result

The desired window is selected and displayed on the popupmenu.

Notes

• The windows are implemented using functions from the Matlab Signal Processing Toolkit which is required to use this dodad.

• This control is only available when the current filtering method is set to FIR.



Order Popupmenu

Purpose

Sets the filter order.

Use

1. Place the cursor on the Order popupmenu and click the left mouse button once to open the menu.
2. Place the cursor on the desired filter order and click the left mouse button. The popupmenu contains orders of 11, 21, 31, 41, 51 and 61 when the FIR filter is selected and orders of 2, 4, 6, 8, 10 and 12 when an IIR filter is selected. If the desired order is not among these options, the required order can be entered manually by selecting the "User" option. In such a case, an edit box appears in place of the popupmenu. Enter the desired order into the edit box and press return.

Result

The filter order is set.

Notes

• The filter order for bandpass and stopband filters is twice the order shown on the popupmenu. This is consistent with the input argument specifications for the Mat-

lab Signal Processing Toolbox routines.

• Highpass and stopband filters requires an even order. Odd orders are automatically rounded up one to the next higher even order.

• The filters are implemented using functions from the Matlab Signal Processing Toolkit which is required to use this dodad. Some restrictions on the filter order apply when using these functions. See the signal processing toolkit documentation for specifics.

• In general, FIR filters require a high order to obtain a sharp rolloff. IIR filters can achieve the same or better results using a much lower order.

Lower Cutoff Frequency Edit Box

Purpose

• Displays the lower cutoff frequency when it is selected using the mouse.

• Allows the operator to manually enter the lower cutoff frequency.

Use

When selecting by mouse:

1. Display the spectrum of the signal using the Spectrum radiobutton.
2. Mark the lower cutoff frequency using the beginning mark.

Manual entry:

1. Place the cursor over the edit box and click the left mouse button.
2. Enter the desired lower cutoff frequency and press return.

Result

The lower cutoff frequency is displayed in the edit box.

Notes

• If the lower cutoff frequency is (1) less then or equal to zero or (2) greater than or equal to $f_s/2$ or the upper cutoff frequency, the error message "Invalid lower cutoff frequency entered. Try again..." is displayed in the Matlab command window and the edit box is cleared. This cycle continues until a valid lower cutoff frequency is entered.

• After a cutoff frequency has been set, the transfer function of the currently defined filter is drawn against the background of the signal spectrum. An unstable filter can generally be recognized from the transfer function.

• Setting the lower cutoff frequency too close to zero, $fs/2$ or the upper cutoff frequency can lead to an unstable filter.

• The effect of an unstable filter is immediately recognizable after applying the filter.

Upper Cutoff Frequency Edit Box

Purpose

• Displays the upper cutoff frequency when it is selected using the mouse.

• Allows the operator to manually enter the upper cutoff frequency.

Use

When selecting by mouse:

1. Display the spectrum of the signal using the Spectrum radiobutton.
2. Mark the upper cutoff frequency using the end mark.

Manual entry:

1. Place the cursor over the edit box and click the left mouse button.
2. Enter the desired upper cutoff frequency and press return.

Result

The upper cutoff frequency is displayed in the edit box.

Notes

- If the upper cutoff frequency is (1) less then or equal to zero or (2) greater than or equal to $f_s/2$ or the lower cutoff frequency, the error message "Invalid upper cutoff frequency entered. Try again..." is displayed in the Matlab command window and the edit box is cleared. This cycle continues until a valid upper cutoff frequency is entered.
- Setting the upper cutoff frequency too close to zero, $f_s/2$ or the lower cutoff frequency can lead to an unstable filter.
- The effect of an unstable filter is immediately recognizable after applying the filter.

Passband Ripple Popupmenu

Purpose

Sets the amount of allowable ripple in the passband of Chebychev or elliptical filters.

Use

1. Place the cursor on the Passband Ripple popupmenu and click the left mouse button once to open the menu.
2. Place the cursor on the desired passband ripple and click the left mouse button. The popupmenu contains ripple values of 0.01, 0.05, 0.1, 0.5, 1.0 and 5.0 decibels. If the desired ripple is not contained amoung the available values, the ripple can be entered manually by selecting the "User" option. In this case, an edit box appears in place of the popupmenu. Enter the desired ripple into the edit box and press return.

Result

The passband ripple is set.

Notes

This control is only available when the current filtering method is set to Chebychev Type 1 or Elliptical.

Stopband Attenuation

Purpose

Sets the amount frequencies in the stopband are attenuated.

Use

1. Place the cursor on the Stopband Attenuation popupmenu and click the left mouse button once to open the menu.
2. Place the cursor on the desired stopband attenuation and click the left mouse button. The popupmenu contains attenuation values of 10, 15, 20, 30, 40 and 50 dB decibels. If the desired attenuation is not contained amount the available values, the attenuation can be entered manually by selecting the "User" option. In this case, an edit box appears in place of the popupmenu. Enter the desired attenuation into the edit box and press return.

Result

The stopband attenuation is set.

Notes

This control is only available when the current filtering method is set to Chebychev Type 2 or Elliptical.

Linear/Logarithmic Radiobutton

Purpose

Shows the power spectral density display scaled linearly, $|H(f)|^2$, or logarithmically in decibels, $20 \log H(f)$.

Use

1. Place the cursor on the desired radiobutton option and click the left mouse button.

Result

The display is redrawn appropriately.

Notes

This control is available only when the Spectrum radiobutton is active.

Apply Pushbutton

Purpose

Applies the filter.

Use

1. Click the left mouse button while the cursor is over the Apply pushbutton.

Result

The filter specified by the other controls is applied to the vector and the result is displayed.

Notes

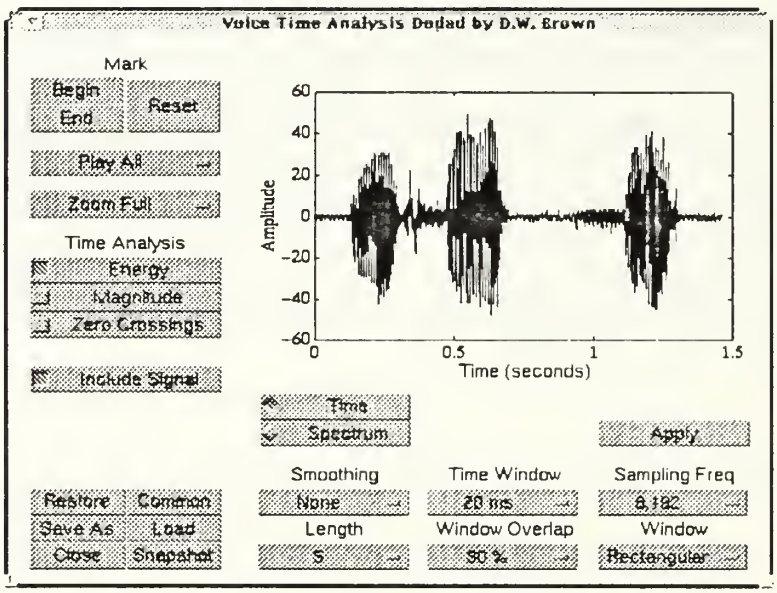
The result of an unstable filter is immediately recognizable. Selecting Restore returns the original signal.

VECTTIME

The Speech Time-Domain Analysis Dodad provides and graphical, interactive environment to apply time-domain methods of determining voice and unvoiced phonemes in speech. Available time-domain analysis algorithms are the short-time energy and magnitude methods for determining voiced phonemes and the short-time zero crossings method for determining unvoiced phonemes. Voiced phonemes are characterized by a relatively high energy content while unvoiced phonemes are characterized by higher frequency content (hence, a higher number of zero crossings).

Starting the Speech Time-Domain Analysis Dodad

The Speech Time-Domain Analysis Dodad can be started in one of two different ways. The first is to start the dodad with a vector name as an argument. A vector is defined as a 1xN or a Nx1 Matlab variable and it must exist in the Matlab workspace before starting the Speech Time-Domain Analysis Dodad. To analyze a vector named "myvector", the command "vecttime(myvector)" needs to be executed.



No output arguments are supported. Speech time-domain analysis curves must be saved using the Save pushbutton described below.

The second way to start the Speech Time-Domain Analysis Dodad is to execute the "vecttime" command without any arguments. This can be done only after the vectedit or vectfilt dodad has been used to modify a vector. Every time a vector is modified using these dodads, the modified vector is saved to a common vector in the global area of the Matlab workspace. When no vector argument is supplied, this common vector is loaded into the Speech Time-Domain Analysis Dodad.

Speech Time-Domain Analysis Dodad Specific Controls

In addition to controls discussed under *common controls*, the following controls are specific to the Speech Time-Domain Analysis Dodad (vecttime). These controls setup the desired analysis method, the analysis frame length and overlap, and output curve smoothing options.

Time Analysis Checkboxes

Purpose

Selects the analysis method to be performed.

Use

1. Place the cursor over the desired analysis and click the left mouse button to turn the analysis method on or off.

Result

The desired is performed when the Apply pushbutton is selected.

Include Signal Checkbox

Purpose

Includes a plot of the time-domain signal in the output graphic window.

Use

1. Place the cursor over the Include Signal checkbox and click the left mouse button to turn the inclusion option on or off.

Result

The time-domain signal is included or excluded when the Apply pushbutton is selected.

Smoothing Popupmenu

Purpose

Selects the smoothing method to be applied to the output analysis curves, if any.

Use

1. Place the cursor on the Smoothing popupmenu and click the left mouse button once to open.
2. Place the cursor on the desired smoothing method (None, Average or Median) and click the left mouse button.

Result

The desired smoothing method is selected and displayed on the popupmenu.

Smoothing Filter Length Popupmenu

Purpose

Selects the length of the smoothing filter employed on the output analysis curves.

Use

1. Place the cursor on the Length popupmenu and click the left mouse button once to open.
2. Place the cursor on the desired length (3, 5, 7, 9, 11, or 13) and click the left mouse button. If the length is not one of those available, the desired value can be entered by selecting the "User" menu item. In this case, an edit box appears in place of the popupmenu. Enter the desired length into this edit box and press return.

Result

The desired length is selected and displayed on the popupmenu.

Time Window Frame Length Popupmenu

Purpose

Selects the length of the analysis window frame.

Use

1. Place the cursor on the Time Window popupmenu and click the left mouse button once to open.
2. Place the cursor on the desired length (5 ms, 10 ms, 15 ms, 20 ms, 30 ms, or 50 ms) and click the left mouse button. If the length is not one of those available, the desired value can be entered by selecting the "User" menu item. In this case, an edit box appears in place of the popupmenu. Enter the desired length in milliseconds into this edit box and press return.

Result

The desired length is selected and displayed on the popupmenu.

Time Window Overlap Popupmenu

Purpose

Selects the length of overlap of consecutive analysis windows.

Use

1. Place the cursor on the Window Overlap popupmenu and click the left mouse button once to open.
2. Place the cursor on the desired overlap (10%, 20%, 30%, 50%, 75% or 90%) and click the left mouse button. If the desired overlap is not one of the available default choices, the desired value can be entered by selecting the "User" menu item. In such a case, an edit box appears in place of the popupmenu. Enter the desired overlap into this edit box and press return.

Result

The desired overlap is selected and displayed on the popupmenu.

Apply Pushbutton

Purpose

Applies the analysis.

Use

1. Click the left mouse button while the cursor is over the Apply pushbutton.

Result

The analysis method is applied as specified. A graphic window opens to display the results.

Save Pushbutton

Purpose

Saves the short-time energy, magnitude and zero-crossing curves to the Matlab workspace, if the analysis was generated.

Use

1. Place the cursor on the Save Pushbutton and click the left mouse button.
2. The Save and Load pushbuttons are replaced by an edit box. If save has already

been used, the previously entered vector name are contained in the edit box.
3. Place the cursor on the edit box and click the left mouse button. Using the keyboard, backspace over any previous entry and then type in the desired vector name (following standard Matlab variable name conventions).
4. Press the return key.

Result

The data is saved into an Nx4 matrix where the first column is the time indices, the second column is the short-time energy, the third column is the short-time magnitude and the fourth column is the short-time zero-crossings.

WPERIGRM

Purpose

Display the periodogram of a signal using a linear scale (relative magnitude).

Synopsis

```
wperigrm(x)
wperigrm(x,fs)
wperigrm(x,fs,'phase')
```

Description

wperigrm(x) - Computes the periodogram of *x* and displays the result using a linear (relative magnitude) scale. Only the positive frequencies up to half the sampling frequency are displayed. The default sampling frequency is 8192 Hz.

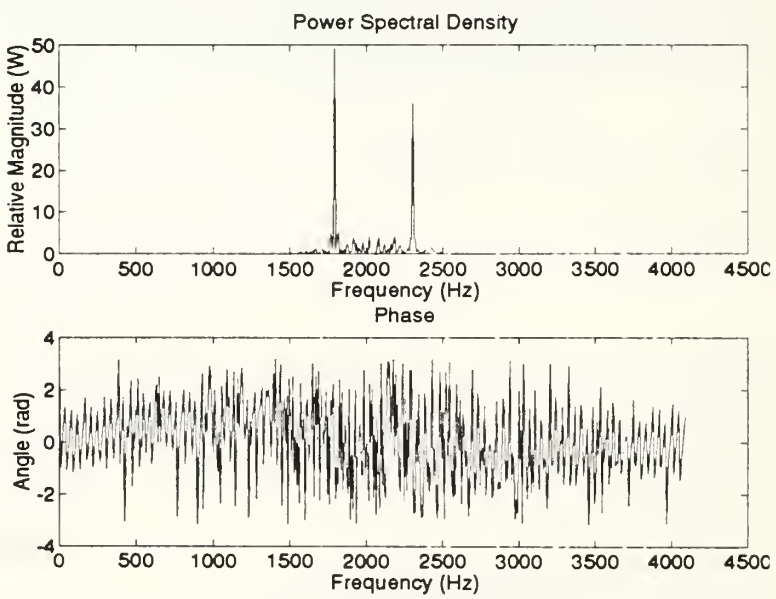
wperigrm(x,fs) - Sets the sampling frequency to *fs*.

wperigrm(x,fs,'phase') - Splits the graphics window and displays both the magnitude and phase information.

Example

Generate a frequency-shift keyed signal and display it's periodogram.

```
y = bfsk(512,512,2048,0.1);
wperigrm(y,'phase');;
```



Algorithm

The periodogram is given by:

$$S_{x_T}(t,f) \equiv \frac{1}{T}|X_T(t,f)|^2$$

where $X_T(t,f)$ is the fourier transform of $x(t)$:

$$X_T(t,f) = \int_{t-T/2}^{t+T/2} x(u)e^{-j2\pi u}du$$

Limitations

The size of the FFT used to compute the fourier transform is the closest power-of-two greater-than or equal-to the length of x. Long input vectors require longer compute times. If x is short and a finer frequency resolution is desired, the input vector x can be zero padded before calling the function.

```
wperigrm([x zeros(300,1)]);
```

See Also

lperigrm, plottime

Reference

[1] William A. Gardner, *Statistical Spectral Analysis, A Nonprobabilistic Theory*, pp. 5-7, Prentice-Hall, 1988.

INITIAL DISTRIBUTION LIST

		No. Copies
1	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School 833 Dyer Road, Room 437 Monterey, CA 93943-5121	1
4.	Professor Monique P. Fargues, Code EC/Fa Department of Electrical and Computer Engineering Naval Postgraduate School 833 Dyer Road, Room 437 Monterey, CA 93943-5121	4
5.	LT Dennis W. Brown c/o Professor Monique P. Fargues, Code EC/Fa Department of Electrical and Computer Enginnering Naval Postgraduate School 833 Dyer Road, Room 437 Monterey, CA 93943-5121	3
6.	LT Dennis Vandenberg, Code 9120 Naval Research Laboratory 4555 Overlook Avenue SW Washington DC 20375	1
7.	Dr. John Lott Naval Security Group Support Activity 3801 Nebraska Avenue NW Washington DC 20390	1
8.	Dr. Cliff Comisky NCCOSC RDTE Division, Code 7701 San Diego, CA 92152-5000	1